National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Services des thèses canadiennes

Ottawa, Canada
K1A 0N4

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Canada

NL-339 (r. 86/06)

The University of Alberta

INFERENCE IN A TOPICALLY ORGANIZED SEMANTIC NET

by

Johannes de Haan

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
Spring, 1986

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-30140-6

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Johannes de Haan

TITLE OF THESIS: Inference in a Topically Organized Semantic Net

DEGREE FOR WHICH THIS THESIS WAS PRESENTED: Master of Science

YEAR THIS DEGREE GRANTED: 1986

Permission is hereby granted to The University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(Signed) .....Johannes de Haan.....

Permanent Address:
9812 88 Avenue
Edmonton, Alberta
Canada T6E 2R3

Dated March 27, 1986

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Inference in a Topically Organized Semantic Net** submitted by **Johannes de Haan** in partial fulfilment of the requirements for the degree of **Master of Science.**

L K Schubert
.....

Supervisor

.....

.....

.....

March 27, 1986

ABSTRACT

A semantic net system in which knowledge is topically organized around concepts has been under development at the University for some time. The system is capable of automatic topical classification and insertion of modal logic input sentences, concept and topic oriented retrieval, and property inheritance of a general sort. It is envisioned that this semantic net will be used as the knowledge base for a natural language conversational system. Throughout the course of a conversation, inference will be needed during input translation, assimilation, question-answering, and to recognize and reason about the plans and goals of the conversational partner.

A method to support some of these inferences is proposed. It is a deductive, resolution based system, enhanced by a set of special inference methods, and builds upon ideas used in the original design of the semantic net. Specifically, the method is able to quickly prove or disprove sentences expressed in the first order predicate logic, using as premises the complete set of sentences in the semantic net.

Acknowledgements

I would like to thank my supervisor, Len Schubert, for the guidance and criticism he provided throughout the work for this thesis. I also appreciate the useful and thought-provoking comments from the members of my examining committee: Renee Elio, Lisa Marckworth-Stanford, and Jeff Pelletier.

The financial support from the Natural Sciences and Engineering Research Council of Canada and from the Department of Computing Science is gratefully acknowledged.

I am indebted to Mildred Thill, Leo Hartman, and Mary-Angela Papalaskaris for the various forms of help and moral support they have given me.

Table of Contents

Chapter	Page
Chapter 1: Introduction	1
1.1. Objectives and scope	3
1.2. Overview	5
Chapter 2: Inference for question-answering	6
2.1. Non-deductive inference	6
2.2. Deductive inference	12
2.2.1. Knowledge representation in these systems	13
2.2.2. Knowledge organization in these systems	14
2.3. Inference and control of inference	14
2.4. Controlling deduction	16
Chapter 3: The Semantic Net	20
3.1. Introduction	20
3.2. Representation of knowledge in the net	20
3.3. Organization of knowledge in the net	23
Chapter 4: Approach	31
4.1. Introduction	31
4.2. An overview of the method	31
4.3. Inference with the representation	32
4.4. Resolution	34
4.5. Enhancement of resolution	36
4.6. Control of resolution	37

4.6.1. Syntactic control 38

4.6.2. Semantic control 39

4.6.3. Other control strategies 41

Chapter 5: Implementation 45

5.1. Description of the algorithm 45

5.2. QA control structures 45

5.3. Checkpointing the net 45

5.4. Concurrent proof and disproof 47

5.5. Entering a clause 47

5.6. Classification of question clauses 48

5.7. Ranking access points 52

5.8. Ranking possible resolving actions 54

5.9. Unification and resolving 54

5.10. Implementation details 55

Chapter 6: Conclusions 56

6.1. Future research 57

References 61

Appendix A: Sample Output 63

A.1. Questions and timing 63

A.2. The LRRH knowledge base 64

A.3. Access skeltons for the knowledge base 68

A.4. Tracing the QA system 71

List of Figures

Figure		Page
2.1	The 'diplomatic meetings' E-MOP	8
2.2	The 'borrow' MOP and 'favor' META-MOP	10
3.1	Semantic net for the clauses C1, C2 and C3	23
3.2	A topical hierarchy (TH) for physical objects	24
3.3	Topic access skeleton (TAS)	25
3.4	A concept hierarchy (CH) for physical objects	27
3.5	Concept access skeleton (CAS)	28
4.1	Connection graph for the GM problem	42
4.2	Graph for the GM problem after three resolutions	43
5.1	Pseudo-code for the QA algorithm	46
5.2	Question classifications	50

Chapter 1

Introduction

A semantic net system in which knowledge is topically organized around concepts has been under development at the University for some time [CoS80,Sch76,SGC79]. The system is capable of automatic topical classification and insertion of modal logic input sentences, concept and topic oriented retrieval, and property inheritance of a general sort. More recently, the net has been supplemented by special representations and inference methods for types, parts, colours, and time [SPT83].

It is envisioned that this semantic net will be used as the knowledge base for a natural language conversational system. To properly understand what is said during the course of a typical conversation, a variety of inferences using knowledge in the net will need to be made:

1. During input translation inference will be needed to decide among syntactic alternatives. For example, inference based on knowledge about the world is needed to correctly determine that in the sentence "The woodcutter killed the wolf with the axe", the prepositional phrase "with the axe" modifies the verb "kill", whereas in "The woodcutter killed the wolf with the big ears", the prepositional phrase "with the big ears" modifies the noun "wolf". Similarly, world knowledge is needed to decide among semantic alternatives. For example, the verb "took" should be interpreted in one way in "Little Red Riding Hood took the basket to Grandma's", and another in "Little Red Riding Hood took the path to Grandma's"

2. Inference is needed during the assimilation of input sentences. For example, in a conversation, references to objects or persons can appear in many forms (most commonly by the replacement of nouns by pronouns) and knowledge about the conversational context is needed to correctly relate these references to known concepts. The

proper understanding of sentences also includes the ability to infer consequences and implications, and to realize when newly acquired knowledge contradicts previous knowledge. For example, consider the following two sentences: "The door burst open. It was the woodcutter." Inference is needed to properly connect them, and to realize that the woodcutter is probably now 'inside'. Not all implications need be that complex – they can be as simple as realizing that if A is inside B, then A is physically smaller than B, and that B is a container or enclosure of some sort.

3. Obviously, inference is needed to generate answers to direct questions posed by the conversational partner. This is not restricted to merely deciding between 'yes' or 'no', but must include the ability to generate complete answers, and to provide more information if necessary.

4. The ability to provide complete answers to questions in turn implies the ability to recognize and reason about plans and goals that the conversational partner is pursuing. More generally, inference is needed to plan conversational behavior throughout the discourse.

This thesis describes a method which can be used to support some of these inferences, using the complete semantic net as its knowledge base. It is a deductive, resolution based system, enhanced by a set of special inference methods, and builds upon ideas used in the original design of the semantic net. The method has been successfully implemented as a PASCAL program.

1.1. Objectives and scope

Some of the inferences described in the previous section are rather sophisticated, and it is beyond the scope of this thesis to explore them. However, the ability to make any of these inferences depends on access to knowledge embodied in the semantic net. Similarly, methods for these kinds of inference will all rely on some basic inferential ability to use this knowledge, in a way that exploits the organization and structure of the net. It is this basic level that the inference method described in this thesis implements - the ability to quickly prove or disprove facts that logically follow from knowledge in the net, without getting 'bogged down' by the large amount of knowledge stored in the net. Furthermore, to meet part of the overall goal of natural language understanding, it is able to make these inferences in times much less than a second, and for a wide variety of topics.

Specifically, the method is able to quickly prove or disprove sentences expressed in the first order predicate logic, using as premises the complete set of sentences in the semantic net. Given a sentence, it attempts to simultaneously prove and disprove the sentence, and provides a corresponding 'yes' or 'no' answer if it succeeds. Alternatively, if it cannot find either a proof or disproof, due to a lack of knowledge or because it exceeded the limit on some resource, it will answer, "don't know". In this respect, it falls within the definition of a theorem-prover: the goal is to prove or disprove the validity of a group of statements expressed in the language of some system (usually mathematical or logical), using rules of inference defined for that system. However, the domain in which this method operates (natural language) is different from traditional theorem-proving domains, and there exist several fundamental differences:

1. Size of the knowledge base.

   Theorem provers work on problems in very well defined logical or mathematical domains, the kind of problems introductory students in these disciplines are often asked

to do. The systems are artificial and are usually axiomized by some small set of statements, so that the total body of knowledge the theorem prover works with is relatively small. This is not true of the semantic net, which will incorporate a very large body of knowledge, sufficient at least to carry on an intelligent conversation. It is tempting to try to split the problem into two: retrieve a subset of the knowledge base, and run a theorem-prover on the subset. However, inference is needed to retrieve just the right subset needed to solve the problem, and this 'solution' just begs the question.

2. Deductive ability.

Theorem-provers are judged by their mathematical or logical deductive ability - better provers solve logically more complex problems. People require minutes or even hours to solve these same problems, and yet, they are able to perform the inference needed for natural language understanding almost instantaneously. It is not unreasonable to suppose that natural language inference is shallower (i.e., requires fewer steps) than that required for mathematical theorem proving. Further psychological support is provided by asking people to explain the reasoning which lead them to make an particular inference. Explanations for inferences made during natural language processing are usually short, and contain a small number of steps: "I knew the wolf wasn't killed using a set of ears, because ears are not usually lethal weapons. I also knew that this particular wolf had big ears." In contrast, explanations for answers to logical conundrums are expressed as laborious chains of inference, which most people cannot even follow verbally. All this is not to say that a mechanical method for natural language understanding has to model people - it only has to emulate them successfully, but using human behavior as a guide is certainly a good first step.

So, whereas the goal for theorem-provers is to find methods that successfully find longer and more complicated inference chains, the goal for inference for natural language understanding is to find representational, organizational and inferential

systems such that the number of steps required to make an inference remains small. For example, semantic networks organize knowledge about a concept so that it is clustered around the concept, and a minimum amount of inference is necessary to collect that knowledge when it is needed. Or, special inference methods for types are used to collapse long traversals in a taxonomic hierarchy into a single, simple step.

3. Theoretical Considerations

A third difference, which is perhaps not so fundamental, is that theoretical considerations are less important for natural language understanding than for theorem-proving. For example, theorem-provers consider it desirable that the inference method they use be provably complete. This is not the case for natural language understanding, where greater emphasis is placed on getting to the right subset of knowledge than on being able to deduce all the possible conclusions from that knowledge.

The method presented in this thesis could be called a yes/no question-answering (QA) system, using a rather narrow definition of that term. To emphasize this function, the method will henceforth be referred to as a QA method, and the system which implements this method will be called the QA system.

1.2. Overview

The rest of this thesis is organized as follows: Chapter 2 surveys related work in this area, including work on theorem proving, knowledge representations (especially semantic networks) which support deduction and natural language inference, and question-answering in general, Chapter 3 describes the semantic net, Chapter 4 describes the approach used by the QA system, Chapter 5 describes details of the implementation, and Chapter 6 presents some conclusions and suggests directions for future research.

Chapter 2

Inference for question-answering

2.1. Non-deductive inference

One of the major goals of any inference system for large bodies of knowledge is to be able to get at just the right subset of the knowledge needed to answer a question. Enumerating each fact in the knowledge base (KB) and asking, "Is this fact relevant to the question?" is the simplest way, but it is linear in time, and worse yet, determining relevance from the surface form of the fact is usually a non-trivial task. Semantic networks reduce the scope of an enumeration by storing related facts near each other, but an amorphous net does not narrow the search enough, and determining relevance is still a problem.

One way of quickly determining relevance is to organize knowledge as it is added to the KB, to categorize it in such a way that when a question is asked, we can say, "All the knowledge in this category (or these categories) is likely to be relevant to the question, and furthermore, knowledge in other categories is not likely to be relevant". By making these decisions, an organizational scheme constrains inference with the knowledge it organizes, and thus implicitly becomes part of the overall inference method. Most organizational systems make these decisions as knowledge is assimilated, and not during the retrieval process. In that sense, they form one of the larger non-deductive components of many inference systems.

For example, CYRUS [Kol81] is a question-answering system which tries to automatically categorize the knowledge it is given. CYRUS stores and retrieves episodes in the lives of former U.S. Secretaries of State Cyrus Vance and Edmund Muskie. It does not make any inferences with the episodic knowledge itself, but it does decide how to index these episodes,

and will also deduce plausible indices for questions that do not contain a direct reference to the indices it has chosen.

CYRUS's memory organization is based on Episodic Memory Organization Packets (E-MOPs), each of which is used to categorize a set of episodes. Each E-MOP has two important aspects - generalized information (called norms) that is common to all the episodes it categorizes, and a tree-like structure that indexes these episodes by their differences. Entries in the index can either be episodes or specialization E-MOPs, which further categorize a particular subset of episodes. Figure 2.1 illustrates the general E-MOP called 'diplomatic meetings' and two of its specialization E-MOPs, for 'Begin' and 'Camp David Accords'. As evident in the figure, events (labeled EV2, EV3, and EV4) can be indexed by more than one E-MOP.

Every new event that is added to CYRUS's memory is first scanned to extract features to be used as indices. A major attribute of these features is their predictive power, as judged by previous experience. CYRUS does not automatically make this judgment, but is does infer a second important property, that of discriminability. This is done by simply discarding those features which already appear as norms on existing E-MOPS, to leave a set of discriminating features.

CYRUS builds new E-MOPS whenever two events with the same features (which do not fit into an existing E-MOP) are detected. The new E-MOP's norms will be those features which are common to both events, and an index is built to reflect the discriminating features of the two. CYRUS has the capability of changing these initial decisions somewhat, either by removing 'bad' norms, or by collapsing a pair of generalization/specialization EMOPs which categorize many of the same events.

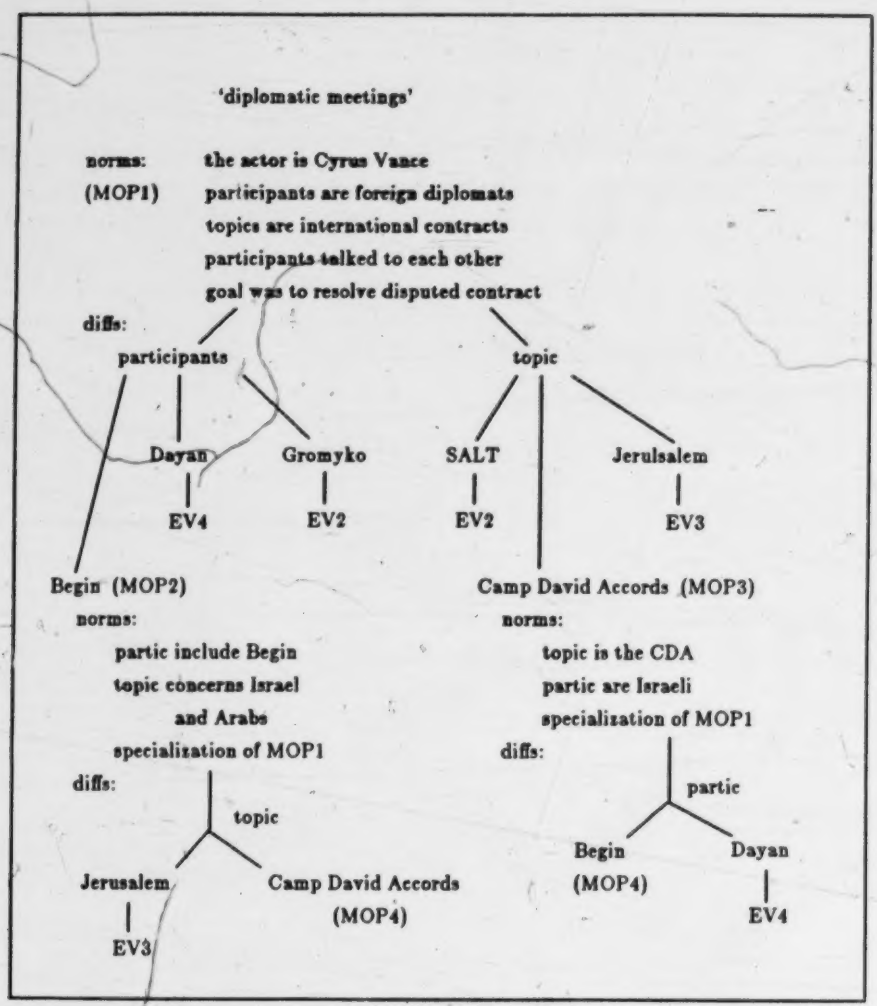


Figure 2.1 The 'diplomatic meetings' E-MOP

To retrieve episodes, CYRUS only has to look at the features indicated by the question and follow an E-MOP hierarchy down to a matching event. The top level E-MOP at which to start is determined by 'component-to-context instantiation' rules. For example, in answering, "Who have you talked to about SALT?", CYRUS uses information about SALT to infer the context of a 'political meeting'. If this initial search fails, CYRUS will try to infer further information not directly contained in the question. If there are no features

specified by the request which can be used to directly find an event, special instantiation rules are used to infer new features. For example, if the request was "Have you ever attended a meeting in Jerulsalem", and 'location' was not judged to be a predicative feature, then an instantiation rule could be used to infer a feature which is predictive, such as the participants of such a meeting (Isreali dignitaries). If the application of these rules still doesn't yield anything, CYRUS will try to use 'context-to-context instantiation' rules to switch to another E-MOP hierarchy, in the hope that events in that E-MOP will refer to the event requested.

Aside from these two kinds of rules, CYRUS does most of its organizational work (or inference) before any questions are asked, and as a result retrieval becomes faster. Similarly, it should be possible to make question-answering faster by doing the same kind of 'forward inferencing' on the knowledge itself.

BORIS [LDJ83] is a question answering system for stories which does exactly that. To understand the kinds of knowledge which appear in stories, BORIS uses high level knowledge structures which try to encapsulate those same kinds of knowledge. The most important of these are: (1) Memory Organization Packets (MOPs), which try to represent knowledge about events, plans, and goals; (2) META-MOPS, which represent abstract concepts not easily expressible in a MOP; and (3) Thematic Affect Units (TAUs), which try to represent knowledge people often express in adages such as "Every cloud has a silver lining". For example, BORIS contains a 'borrowing' MOP which describes the essentials of a borrowing episode, a 'favor' META-MOP which describes what is involved in the process of receiving and returning a favor (i.e., an abstract kind of borrowing), and a 'hidden blessing' TAU for the 'silver lining' adage. TAUs usually involve variations from the standard goal/planning processing handled by MOP or META-MOPs, and try to capture the implications of unexpected events, such as when a plan goes wrong (or almost goes wrong). Figure 2.2 illustrates the 'borrow' MOP and 'favor' META-MOP. Episodic memory in BORIS consists of

fragmentary instantiations of MOPS, META-MOPS, and TAUs, all linked together as one network. Other organization structures can also appear in this net, but the three mentioned are the most important.

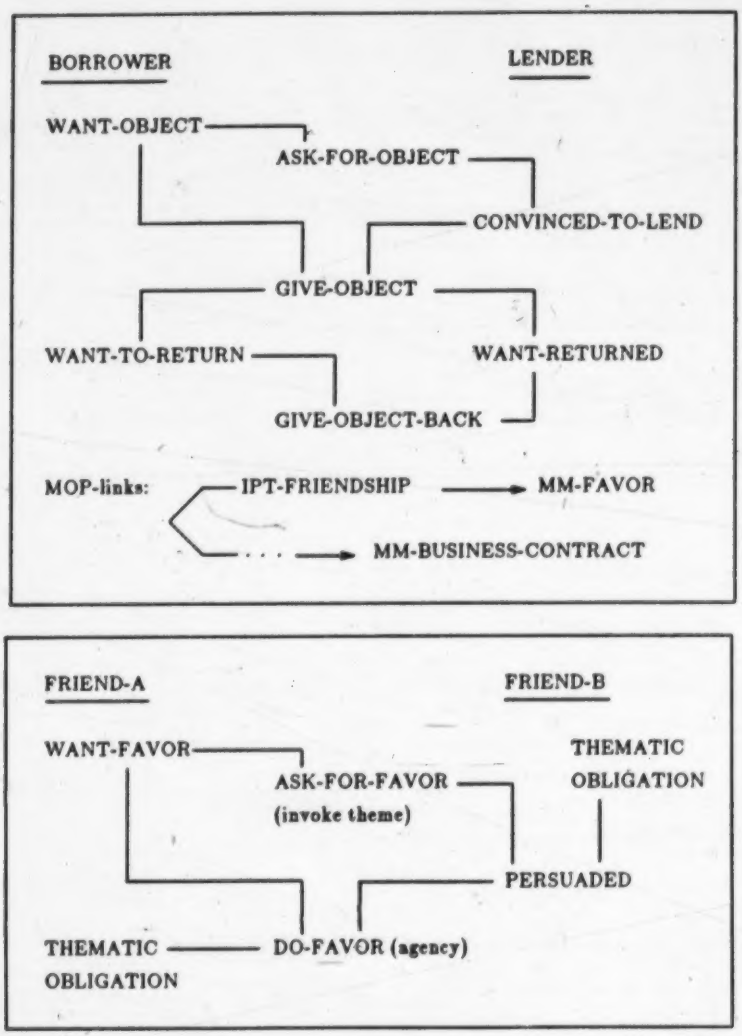


Figure 2.2 The 'borrow' MOP and 'favor' META-MOP

As BORIS 'reads' a story, it starts building an episodic memory for the story. As each

word is read, concepts are placed in memory, and special processes called demons, are spawned. For example, the word 'spill' may create a 'propel' concept in memory, and spawn demons to look for the actor (who spilled) or object (what spilled) for that concept. As the demons fire, corresponding episodes are built or retrieved from episodic memory. Demons fall into five categories: (1) demons which fill slots (as for the object slot of 'propel'); (2) goal/planning demons (which might check if the 'spill' was the fulfillment of some goal); (3) setting-related demons; (4) demons dealing with events; and (5) expectation-violation demons (which might check for possible thematic situations (i.e., TAUs) in case the 'spill' was accidental). This building of episodic memory is BORIS's main inference mechanism.

Not all of BORIS's knowledge is in episodic memory - a large amount of its general world knowledge is declaratively encoded in what is called the semantic memory. Bottom-up knowledge is applied to each new event, goal, or state arising from the next input. For example, goal/plan rules are used to detect key goal situations such as goal conflict and goal completion. Top-down knowledge is applied by searching memories BORIS has already instantiated. For example, the arrival of food is an event which could be matched against one of the expectations of going to a restaurant. Meta-story knowledge about how stories are told can also be used. For example, the identification of the main character can drive expectations based on her relationships with other characters in the story. In most cases, it is the job of demons to use this declarative knowledge to build the appropriate episodic structures.

To answer questions then, all that should be necessary is to match the question against what has been inferred during story understanding (i.e., forward inference) and retrieve the answer without further inference. Earlier versions of BORIS did just that. However, to be able to answer all possible questions, forward inferencing must generate a large amount of knowledge that will likely never be used, and even then it will probably miss some inference that should have been made. To solve this problem, BORIS only partially instantiates

structures in episodic memory during story understanding, and allows further instantiation to occur during question understanding. If forward inferencing has not already derived the answer, then this further deductive inference should. Questions are treated just as assertions (with some minor changes to check pre-suppositions and wh-words), and bindings created during question understanding form the basis for the answer.

2.2. Deductive inference

BORIS's understanding is limited first by the expressive power (or lack thereof) of its representational system (the small set of primitives in Schank's conceptual dependency formalism), and second by its organizational structures (i.e., MOPs or META-MOPS). This helps to constrain the amount of forward inferencing during story understanding, but it also pre-determines exactly what inference can or can't be done. For example, BORIS only understands divorce stories, because building the organizational structures is a painstaking task that has to be done by hand, and structures have only been built for that domain. More importantly, the process of understanding a question is not necessarily equivalent to answering the question, and BORIS cannot make deductions that are not already encoded into its representational or organization structures, irrespective of whether these structures are instantiated during story or question understanding.

It seems reasonable to assume that at least in some cases it would be more appropriate to use a truly deductive inference method, which is only constrained by the knowledge (or lack of knowledge) it must operate with. I will discuss three such deductive inference systems: (1) the K-NET knowledge base and its problem solving system, SNIFFER, described in [FiH77]; (2) the Semantic Network Processing System (SNePS), described in [Sha79]; and (3) The Meta-level Representation System (MRS), described in [Rus85]. None of these systems implements a specific network or inference method, providing instead a developmental

environment for work in these areas (somewhat like the logic programming language PROLOG).

2.2.1. Knowledge representation in these systems

K-NET and MRS use the first order calculus as their representational system, although quantified variables are included in a different manner than in most such systems. Existential quantification is done implicitly by using some uniquely named constant in place of the variable. In K-NET, universal quantification is signaled by the presence of a concept in both the antecedent and consequent of an implication (if a variable is not already present in this manner it can be made so, by changing a formula such as P(x) to ELEMENT_OF(x, universe) => P(x)). SNIFFER is a natural deduction system, and K-NET formulae are stored in the form in which they are given to the system. MRS uses either Modus Ponens or resolution as its main rule of inference. For the former, most knowledge must be expressed as implications, and for the latter, MRS will automatically convert all formulae to conjunctive normal form. MRS has one interesting extension to the predicate logic, in that it allows the specification of default rules. For example, it is possible to say "If it is not provable that a particular bird does not fly, then assume the bird can fly."

Nodes in a SNePS net are divided into three groups: (1) assertions, which are essentially equivalent to ground instances; (2) patterns, which are assertions in which some of the constants are replaced by variables; and (3) deductive rules, which use an expanded set of connectives to form inference rules composed of assertions and patterns. The connectives are powerful enough to include negation, disjunction, conjunction and implication. Universal and existential quantification within the deductive rules is possible. Note, however, that it is not possible to embed rules within each other.

2.2.2. Knowledge organization in these systems

K-NET and MRS provide facilities for partitioning knowledge into subnets. In K-NET, the relationships between the subnets can be explicitly and easily represented. Subnets can be hierarchically embedded in other subnets, and a context mechanism exists to allow only a given set of subnets to be "visible" to the inference mechanism at any one time. Subnets in MRS (called theories) are more rudimentary, serving only to include or exclude specific knowledge groupings. SNePS does not partition its knowledge.

K-NET is the only system of these three that allows a taxonomic organization (of sets). K-NET and SNePS also allow named access to nodes, whereas MRS does not.

2.3. Inference and control of inference

At the lowest level, all three of these systems use some form of pattern matching to answer simple queries. Variables in these queries will be bound (or, instantiated) to constants upon the successful match of a query to an assertion (or set of assertions) in the knowledge base. If simple pattern matching to ground instances does not return anything, then each of these systems will start matching the query against deduction rules. SNIFFER uses a logically complete set of natural deduction rules to reason backward from the question. MRS also uses backward chaining, but since the only inference rule (aside from resolution) it uses is Modus Ponens, it only matches implications. As an alternative, and if propositions in the knowledge base have been transformed into conjunctive normal form, MRS will use resolution as the inference rule, with linear input and set of support control strategies. SNePS matches the query against the explicitly specified set of deductive rules. SNIFFER and SNePS automatically look for 'no' answers by using the negation of the question, whereas in MRS the negation has to be explicitly formulated (which can probably be automatically done using its meta-level control system, discussed later).

To summarize, all of these systems use natural deduction and backward chaining from the question to find either assertions that match the query directly, or match via a set of deductive rules and a consistent set of bindings. SNePS will also do a limited amount of forward chaining, by storing previously deduced results for later use. MRS does unrestricted forward inferencing, but cautions the user to keep the initial set of propositions small.

SNIFFER and MRS allow procedural attachment (i.e., they allow the user to augment the declarative knowledge with special applicative or imperative code for certain predicates or tasks). This expands the inferential ability of these systems and provides a hook into alternative representations.

SNIFFER uses a task tree as its central control structure. Each task operates in a environment which specifies the set of bindings for the query that have already been done. Within that environment, there will then exist a set of binder tasks whose job it is to find bindings for variables that are still free. New bindings generate new environments, and the process continues until all variables have been bound (and the question is answered). Most of the control in SNIFFER is centred about deciding which binder task to do next. To perform this selection, SNIFFER uses heuristics such as: knowledge about the semantics of the binding, effectiveness of the task's search method, number of possibilities to check, etc. The general approach is to first seek bindings that are most highly constrained. It is possible for the user to include guidance, or to supply complete expert binding tasks for specific domains.

SNePS and MRS also use a multi-tasking environment and some sort of task selection method to control their inference. However, MRS has added a meta-level control language which allows the user to very easily implement domain specific inference strategies. For example, it is possible to program MRS to recognize and apply forward chaining in situations where it would in fact be useful. Or, it is possible to indicate that it is preferable to do a certain kind of task before another. As for the instruction to try forward-chaining, these

preferences can be emebedded in conditionals, to allow the user to specify complex control strategies for control of inference, just as she specifies the knowledge at a lower level.

2.4. Controlling deduction

Even though deduction in these systems is guided by both the question and by various and sundry control methods, they still display an amazing ability to generate enormous amounts of irrelevant and useless deductions. One problem seems to be that, even though they are used to control inference with knowledge, these methods are not very knowledgeable themselves. As the designers of MRS realized, it is the application of domain specific meta-knowledge that is most successful.

Another system, which applies meta-knowledge about types, is described in the deductive question-answering system for the semantic net of McSkimin and Minker [McM77]. This system splits the knowledge base into four components: (1) a semantic graph which describes set-theoretic relationships among concepts in the domain; (2) a database of assertions and inference rules; (3) a semantic form space which defines constraints on the arguments of propositions in the database; and (4) a dictionary which places each domain concept into a specified category. The semantic graph divides the domain into a finite number of named subsets. For example, the sets 'human' and 'non-human' could be disjoint subsets of the set 'animal', and the set 'human' could be further divided into the sets 'judge' and 'congress' (the domain in this case covering some aspects of U.S. politics). Subsets of a common superset are not necessarily disjoint - they are allowed to overlap. Using the sets defined by semantic graph and the set operators of union, intersection, and complement, it is possible to define what are called semantic categories. For example, it would be possible to define the semantic category of judges who are also lawyers as the intersection of the 'judge' set with the 'lawyer' set. The simplest semantic categories correspond directly to a subset in

the semantic graph. Semantic categories are the categories used by the dictionary, and the specifications used by the constraints in the semantic form space.

When answering questions, this semantic information is used in three ways: (1) to semantically unify clauses; (2) to perform semantic well-formedness tests; and (3) to chose among deductive alternatives by using counting information about the semantic categories. All of these actions serve to delimit the search space.

Semantic unification ensures that substitutions to allow a given deduction are semantically valid. For example, a constant can only substitute for a variable whose semantic category is equivalent to, or includes its own category. Or, when two variables are to be substituted for one another then the semantic categories of each variable must overlap, and a new category, which is the intersection of the original two will be created. Semantic unification would allow the unification of the clause ~ELEPHANT(x) | GREY(x) with the clause ~GREY(CLYDE) (where CLYDE is an elephant), but would disallow unification with the clause ~GREY(FRANK) (where FRANK is some person).

To ensure that clauses already in the semantic net are indeed semantically well-formed, and to immediately detect queries which are unanswerable because they are not well-formed, each clause that is input to the system is semantically unified against a corresponding form in the semantic form space. If unification succeeds, then the new clause is well-formed.

Counting information about semantic categories can be used to chose the fastest path from a set of alternative deductive paths. For instance, in answering, "Are there any senators from New York or Maryland who support defense interests?", knowing that there are a total of four senators from these two states enables the system to check whether these four senators support defense interests, rather than checking all senators who support defense interests (possibly 100) to see if any are from New York or Maryland.

McSkimin and Minker imposed semantic constraints on top of the predicate calculus, but it is possible to move these constraints right into the logic. Specifically, the semantic categories they have defined correspond quite closely to the sorts of many-sorted logics. The many-sorted calculus discussed by Walther [Wal83] is one such logic. It is notable because it was used to successfully prove the Schubert Steamroller [Wal85], a problem posed by L. Schubert in 1978, but whose solution eluded mechanical theorem provers until 1984.

Many-sorted calculi are usually treated as an extension to the standard (one-sorted) first order predicate calculus. The major additions to the standard calculus are: (1) a sort structure which defines the allowable sorts, and the relationships between them; (2) a mapping of variable, constant, and function symbols to the allowable sorts; (3) the definition of what constitutes well-formed formulae in the logic, by specifying a sort for each argument position of all predicates and functions; and (4) some inference method which maintains the well-formededness of derived formulae. The advantages of these logics are those cited for McSkimin and Minker's system: they reduce the search space considerably, and ill-formed formulae can easily be recognized as such. Walther presents a resolution-like inference method for his calculus that is proved to be sound and complete.

Another such logic, which also solved Schubert's Steamroller is described by Cohn in [Coh83,Coh85]. One of the problems of most of these logics is that polymorphic predicate and functions (predicates or functions that apply to more than one sort) have to be repeated if the sorts they apply to do not fit neatly into the sort structure. Cohn solves this problem by using a sorting function to define how the sort of a predicate or function varies as the sorts of its inputs vary. This function has four possible values: (1) EE indicates the formula is ill-sorted; (2) TT indicates the formula is a tautology (as for the case of Integer(4)); (3) FF indicates the formula is a contradiction; and (4) UU indicates the formula is well formed, but requires inference to deduce its truth value.

An associated problem is that variables in some many-sorted logics need to be quantified with respect to a particular sort, causing needless repetition of formulae which contain polymorphic predicates. Cohn allows the sorts of quantified variables to remain unspecified, and derives the restriction on the sort of a variable from the argument positions of function and predicate symbols that it occupies. Cohn also gives a complete and sound set of inference rules for his logic.

Chapter 3

The Semantic Net

3.1. Introduction

This chapter describe the representational and organizational schema of the semantic net, serving as an introduction to the net and to the notation used to discuss it. An understanding of the net is essential to an understanding of an inference method for knowledge in the net, so the description is quite detailed.

3.2. Representation of knowledge in the net

The representational scheme used by the semantic net is essentially that as described by Schubert in [Sch76], incorporating changes described in [Cov80,SGC79]. The syntax of the net provides for the representation of formulae in higher-order modal logic[1], with constants, functions, existentially and universally quantified variables, and the usual truth function connectives (negation, implication, disjunction and conjunction). For example, the following formulae can be expressed in the net:

F1: [Mary loves John]

F2: A.x E.y [[x person] => [[y person] & [x loves y]]]

F1 is intended to express "Mary loves John." and F2 is intended to express "Everybody loves somebody". The notation for specifying formulae is as follows:

[] encloses a proposition. To improve readability, the proposition's predicate is placed after its first argument.

[1] Although the net is able to represent and organize modal propositions, the present QA system is not capable of inference with this knowledge. Consequently, further discussion about the net will generally be restricted to its non-modal aspects.

A.x, E.y indicate universal and existential quantification, respectively. Lowercase letters near the end of the alphabet are used for quantified variables.

~,=>,|,& indicate negation, implication, disjunction and conjunction, respectively.

(f x) indicates the function 'f' with a single argument, 'x'.

? indicates that a question (in the form of a proposition) follows.

Remaining symbols stand for propositional predicates or arguments. Predicates are written using lowercase letters, whereas constants are capitalized in at least the first letter.

Even though the net has the expressive ability to store propositions in an unrestricted form, various organizational reasons have lead to the adoption of a normalized form for the internal storage of propositions (see [CoS80] for a detailed explanation). Briefly, to achieve the normal form input formulae are transformed into a set of clauses in the conjunctive normal form. Each clause is a disjunctive set of literals, and each literal is either an atom (atomic proposition) or negated atom. Universal quantification is implicit, and existentially quantified variables become skolem functions of zero or more arguments (where the number of arguments corresponds to the number of universals which included the existential in their scope). A skolem function of zero arguments is equivalent to an unnamed constant. A detailed description of this form, and the procedure to translate input formulae into it appears in [Nil80].

The formulae F1 and F2 generate three clauses:

C1: [Mary loves John]

C2: ~[x person] | [x loves (f x)]

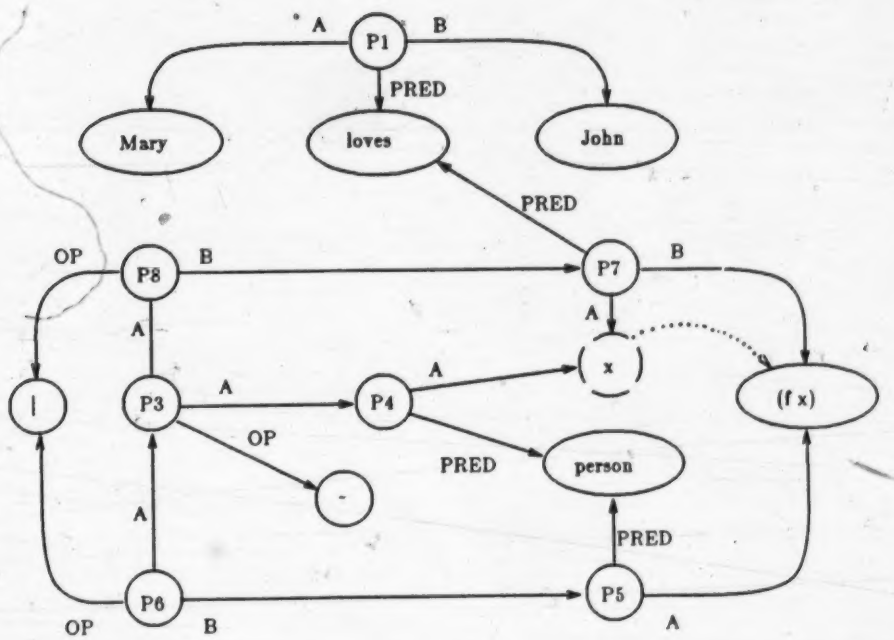
C3: ~[x person] | [(f x) person]

If an input formula is asserted as true, then each of its corresponding clauses is also asserted as true.

In the network, each unique predicate, constant, function and variable is associated with a concept node, and each unique clause, literal and atom is associated with a propositional node. A special class of 'concept' nodes contains the connective operators. The representational structure of the net is then a graph of linked propositional and concept nodes.

In this graph, the propositional node for an atom links the concepts corresponding to the predicate and arguments of that atom. A negated atom is represented by a propositional node that links the 'negation' operator and the propositional node for the unnegated atom. A clause is represented by a propositional node that links the 'disjunction' operator and the propositional nodes for the literals of the clause. When possible, unnecessary propositional nodes are eliminated. For example, a clause that is a single atom will be represented directly by the propositional node for that atom.

Figure 3.1 illustrates the semantic net that represents the clauses C1, C2 and C3. In this figure (which uses the notation of [Sch76]), the dotted arrow indicates that the scope of the universal variable 'x' includes the skolem function generated for the original existential variable 'y'. Every propositional node (labeled Pn) has either a PREDicate link and a set of concept links (labeled A,B,...), or an OPerator link and a set of propositional links. The clause C1 is represented by the propositional node P1, the clause C2 by node P6, and C3 by P8.



C1: [Mary loves John]

C2: ~[x person] | [x loves (f x)]

C3: ~[x person] | [(f x) person]

Figure 3.1 Semantic net for the clauses C1, C2 and C3

3.3. Organization of knowledge in the net

Knowledge in the net is organized around concepts, with topically oriented access to knowledge about each concept. A dictionary provides direct access to named concepts, and each concept in turn points to the complete set of clauses which express some fact about it. The number of clauses about a single concept can be quite large, so a topic hierarchy (TH) is used to organize these clauses. Using the TH, it becomes possible for a retrieval (or inference) method to directly access clauses which topically pertain to a concept and to ignore all others. The structure of the TH also defines relationships among topics, so it is possible to

broaden an access to retrieve clauses which pertain to sub-topics or super-topics of an initial topic. Figure 3.2 presents a topical hierarchy for concepts which are physical objects.

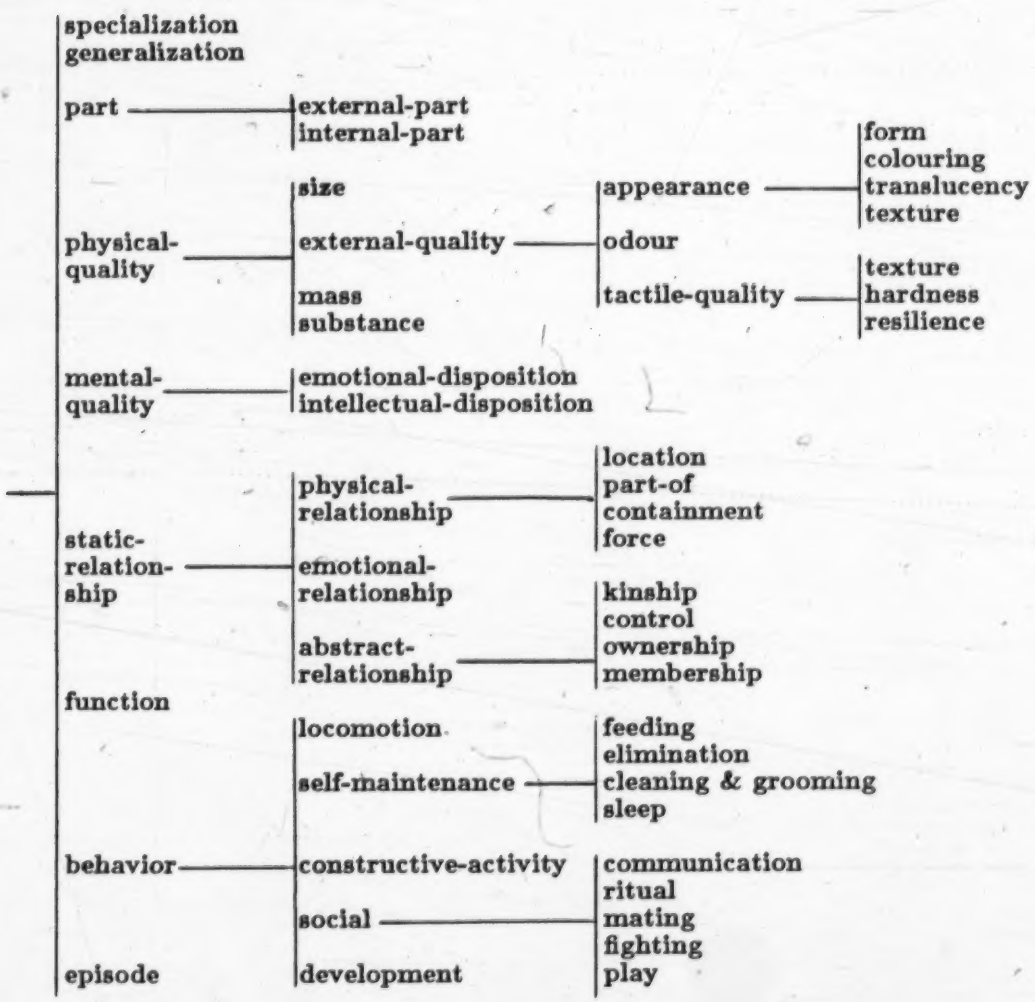


Figure 3.2 A topical hierarchy (TH) for physical objects

For example, the clause A.x [[x wolf] => [x grey]] would be appear under the 'colouring' topic in the hierarchy for the concept 'wolf', and queries about the colouring of wolves would be able to directly access that clause. Queries about the appearance of wolves would

also be able to quickly get to the clause, because 'colouring' is a sub-topic of 'appearance'.

It is not likely that we know something about all topics for all concepts, and to duplicate the entire TH for every concept would waste storage space and increase traversal time across 'empty' topics. To solve this problem, topic access skeletons (TAS) are used. Each TAS is a minimal hierarchy based on the complete topic hierarchy, which only includes topics about which there is some knowledge, or topics which are needed to preserve the structure of the hierarchy. Each concept in the net uses a TAS (which can be regarded as a specific instance of the TH) to organize clauses about that concept. Figure 3.3 presents a TAS which organizes three clauses about wolves. Several detailed examples appear in Appendix A.

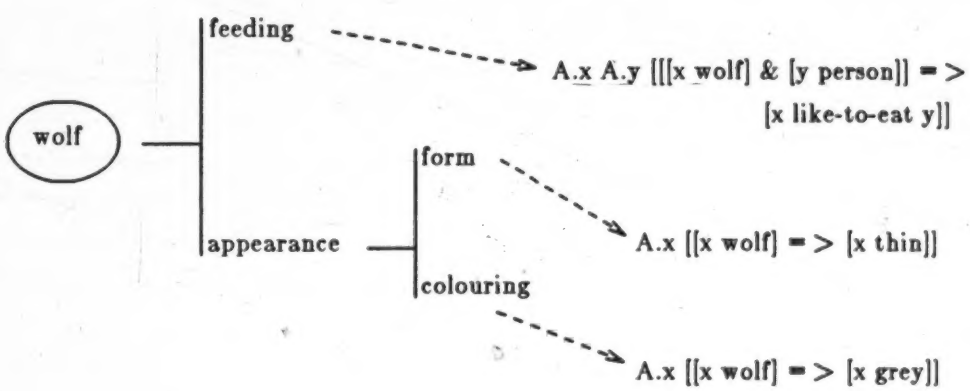


Figure 3.3 Topic access skeleton (TAS)

Just as there is an organization of clauses around a concept, there is an organization of concepts within the entire net. The structure for this organization is called the concept hierarchy (CH). The CH is essentially a type hierarchy for physical objects, defining the relationships between the various types the system knows about. It is used in two ways: (1) for quick associative access to groups of concepts with the same type (just as the TH provides quick access to clauses about the same topic); and (2) to guide a property inheri-

tance mechanism in its search for generalizations of a given concept[2]. Again, to repeat the CH for every subnet is a waste of resources, so a concept access skeleton (CAS) is used instead[3]. Just as a TAS is a minimal hierarchy based on the TH, a CAS is a minimal hierarchy based on the CH. Figure 3.4 presents a CH for physical objects, and Figure 3.5 presents a CAS which associatively organizes some of the concepts in story of Little Red Riding Hood[4] (LRRH). A detailed example appears in Appendix A.

A generalization/specialization hierarchy can easily be used for property inheritance among its constituent concepts (e.g., as from all birds to all thrushes), but it is not quite so easy to use it for property inheritance among Skolem functions representing parts of those concepts. For example, in A.x [[x bird] => [[(f x) head-of x] & ...]] and A.y [[y thrush] => [[(g y) head-of y] & ...]], the Skolem function 'g' (which represents the head-of a generic thrush) is a specialization of the Skolem function 'f' (which is the head-of a generic bird), and should inherit properties from 'f'. The implementation described in [Cov80] included a solution to this problem. First, it required that functions which referred to the same concept be named the same, regardless of what level in the generalization/specialization hierarchy they appeared (e.g., in the above example, 'g' would become 'f'). Then, using Skolem function tables attached to concepts which appeared as arguments of these functions (i.e., the universal variables), it becomes possible to access properties that a specialization such as 'head-of thrush' should inherit from 'head-of bird', by using the function name for 'head-of thrush' as an index into the table for bird, and from there

---

[2] Technically, generalization and specialization relationships specified in the CH are not logically binding. However, the special inference method for types currently uses the CH as its representational structure, so in this implementation the CH exactly mirrors the logical structure.

[3] The net represents modal contexts in subnets, and maintains a separate concept organization for each of them. Since the QA method does not make inferences with modal propositions, in this case there is only one net, and one CAS.

[4] Most examples in this thesis are loosely based on this story.

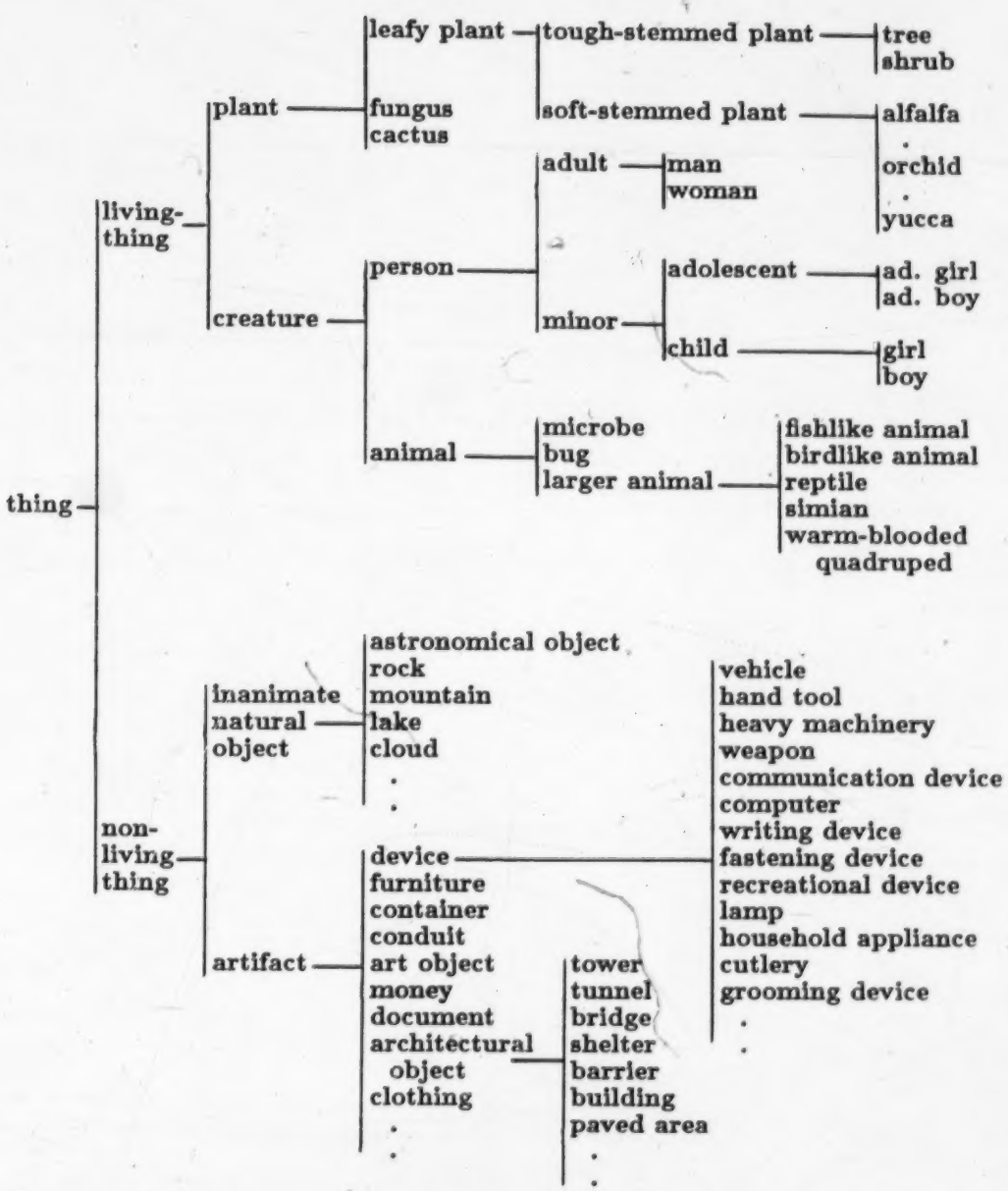


Figure 3.4 A concept hierarchy (CH) for physical objects

quickly getting to the function which represents the 'head-of bird'. This scheme has not yet been implemented in the current system, and property inheritance for Skolem functions

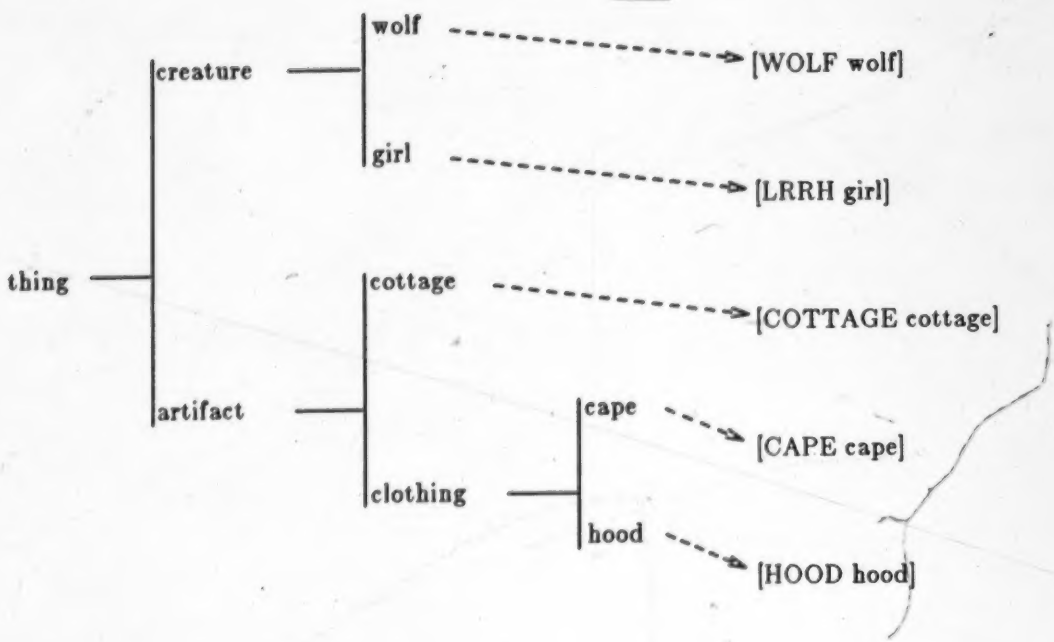


Figure 3.5 Concept access skeleton (CAS)

remains a problem.

Classification of a clause is the process of automatically finding which access skeletons (TAS and/or CAS) should contain pointers to the clause, and where within the skeletons (i.e., which topic or type) the pointers should be maintained. Clauses are classified using a set of pre-defined predicate indicators, which place each predicate into a topical category. For example, the predicates 'red' and 'grey' indicate the topic 'colouring'. Using these indicators, the system determines what combinations of predicate and arguments refer to a particular concept/topic pair. If the concept is a instance (i.e., a constant) and the topic is 'generalization' (type), then the clause is deemed to be a type clause, and is classified associatively in the net's CAS. For example, the predicate 'girl' indicates generalization, so the clause [LRRH girl] would appear in the CAS.

An explanation of the notation used to denote specific access skeleton nodes follows. If a clause is classified under a topic 't' for concept 'c', then the access skeleton node which corresponds to the classification is denoted by 'c−>t'. If a clause is classified under a type 't' in the CAS of the net, then the access skeleton node which corresponds to the classification is denoted by 'NET−>t'. Actually, unless clauses are asserted, the CAS or TAS nodes indicated by the classification do not necessarily exist. In that case, the notation 'c−>t' should be taken to indicate an 'access point', rather than a skeletal node (this distinction will become more relevant when we talk about question clauses, which are not asserted at all classifications). There are four basic classifications for assertions:

1. Topical classification occurs for instances and for more general concepts. For example, [COTTAGE white] is classified under COTTAGE−>colouring and A.x [[x wolf] => [x grey]][5] is classified under wolf−>colouring. When topical classifications are asserted, they are maintained in a TAS off the concept.

2. Associative classification occurs for type clauses of instances. For example, [LRRH girl] would be classified under NET−>girl. When associative classifications are asserted, they are maintained in the CAS of the net. Note that type predicates (called generalization predicates) are also topically classified under the concept, so [LRRH girl] is also classified under LRRH−>generalization.

3. Exclusion classification occurs for propositions that make some statement about about a disjoint relationship between two general concepts. For example, A.x [[x fat] => ~[x thin]] is an exclusion proposition that would be classified under fat−>exclusion and thin−>exclusion.

---

[5] To improve readability, clauses (especially implications) will sometimes be expressed in the original input form.

4. Major implications are implications that arise from accepted models of the world, which humans would be expected to make 'automatically'. For example, the proposition A.x A.y [[x in y] => [x smaller-than y]] is a major implication of 'in', and is classified under in->majorimp. Major implications are the only classification the semantic net does not derive itself, and each major implication must be explicitly indicated as such.

Chapter 4

Approach

4.1. Introduction

This chapter presents a high level description of the QA method (a complete and detailed description of the implementation appears in Chapter 5). It presents the method by which knowledge in the net is used to answer questions, explaining why this particular method was chosen, and occasionally comparing it to other methods.

4.2. An overview of the method

As has been stated, the QA method uses the representational scheme of the first order predicate calculus, and a single rule of inference, called resolution. Factoring is also used, to quickly remove redundant literals from a clause, and because resolution is not compete without factoring. However, equality and paramodulation (the associated resolution method for inference with equality) are not implemented. This has two effects: (1) the expressive power of the system is reduced, as there is no way of saying 'A' is identical to 'B' without renaming one of them; and (2) the usual combinatorical explosion of clauses generated by the use of paramodulation is avoided (although the resolution control methods used by the QA method may also be applicable to paramodulation). Searches for proof and disproof are performed concurrently, but the proofs themselves are logically distinct. Several strategies are used to control or enhance the resolution proof/disproof attempt:

1. Special inference methods are used within the resolution process, reducing long resolution chains to a single step by quickly inferring equivalent, disjoint, or subsumptive relationships between predicates.

2. The QA system routinely tries to evaluate clauses and literals, to arrive at an answer immediately, or to enable a literal or clause to be removed from consideration. Special inference methods which evaluate certain classes of literals are used as adjuncts to the general method.

3. Two traditional resolution strategies, set of support and preference for least complex resolvents, are also used to control the proof attempt.

4. The QA system makes extensive use of the net, both to store intermediate clauses, and, more importantly, to look for and restrict possible resolving actions.

4.3. Inference with the representation

This section discusses some of the implications the choice of the representational system has for inference within that system. In general, the choice of a representational system is not exclusively guided by the potential efficiency of inference methods for that system: other considerations such as expressive power and ease of indexing also play important roles. In particular, the choice of a normalized predicate calculus form was made before work on the QA method started. The net's designers chose the form with an awareness of the implications it would have on inference, but they could not know 'a priori' that an inference method using this form would be sufficient, so the QA system of this thesis is a partial test of their assumptions.

The predicate logic cannot easily express (or is incapable of expressing) several kinds of knowledge that humans use regularly, such as the notions of non-monotonicity (e.g., to infer a particular bird flies because most birds fly, but then to later find out that in fact the bird cannot fly), vague and uncertain knowledge, normalcy, or causality. Inference with these kinds of knowledge requires extensions to the logic. The literature is rife with proposals

to do just that, but they are still embryonic, and further work is needed to determine just what difficulties or complications their adoption entails.

One approach to this problem has been to abandon formal systems such as the predicate calculus altogether. Instead, knowledge is represented at a very high level (such Schank's conceptual dependencies) and some of the burden of solving representational problems is moved onto the inference method. To give a specific example, a common approach to express normalcy is to take advantage of the way an inference method gathers knowledge as it moves up a property inheritance path. If it is known that the inference method stops gathering knowledge about a topic at the lowest level in which it finds anything about that topic, then it would be possible to store "All birds fly" at the generic bird concept, and overrule that by storing "This bird doesn't fly" (perhaps because its wing is broken) at a particular instance of a bird. This system would then infer that particular birds about which nothing explicit is known about their flying ability can indeed fly, and that the bird with the broken wing can't fly. The biggest advantage of this system is that inference about the flying ability of both kinds of birds remains shallow. However, one drawback is that the system is logically inconsistent - it will assert that all birds can fly even when it knows about a bird that can't. Logical inconsistency isn't necessarily all that bad (some humans have been known to be inconsistent at times), but a more severe drawback is that the semantics of the system have been moved into the computer code of the inference method, and are no longer readily available (for example, this system could not explain why it is inconsistent, because it can't examine what caused it to make the conflicting inferences).

Other researchers [FaH85,Pat85] agree that formal systems are desirable, but that not enough emphasis has been placed on the inferential component of these systems. For example, they argue that a logic used by a machine should at least be decidable.

This QA method does use the predicate calculus, in the belief that methods for the standard calculus will still be applicable to an extended calculus. The semantics of the representational and inferential systems are well defined in a formal sense (though the special inference representations and methods complicate that somewhat), and the QA method is logically consistent (although it is probably not complete). However, changes to the representational system are contemplated (it has already been extended to accept modal propositions), and it is likely that other, higher level systems will be used at different levels of the complete natural language understanding system.

The selection of a normalized clause form also has had a significant impact on the design of the QA method. The biggest advantage of the clausal form is that the knowledge base is represented in a simple and consistent manner. This in turn allows storage sharing between clauses with identical literals or atoms, makes the classification process simpler, and provides a clean interface to special inference methods. It also provides a very simple form of inference - that of very quickly determining if a fact (or the negation of a fact) is already in the knowledge base, regardless of the input forms of these facts. However, it does preclude using inference methods which rely in part on the retention of the surface form (such as natural deduction) as a basis for the QA method.

4.4. Resolution

Resolution is a refutational inference method developed especially for machine theorem proving. It was first described by [Rob65] in the sixties, and has since been the basis for very many theorem provers. Consequently, a great deal of work has been done on analyzing and refining resolution proof techniques. A good description of resolution and some of its refinements appears in [Nil80]

The best reason for using resolution is that it is simple to use. A single rule of infer-

ence, called resolution, is sufficient to prove the validity or unsatisfiability of a set of clauses (limited of course by the general undecidability of the first-order predicate logic).

The best reason for not using resolution is that it is not simple to control. To begin with, the number of possible resolutions within the initial set of clauses is usually quite large. Each inference adds another eligible clause and usually many more possible resolutions. Performing a single resolution takes so little time that the number of clauses in an unguided resolution proof attempt grows very quickly, and the number of possible resolutions grows even faster. This 'combinatorial explosion' of clauses makes it very hard to find just the 'right' clauses, and at this point most theorem provers break down, making hundreds or thousands of ineffectual resolutions, and never finding a solution.

Even though the explosion problem has been around for over twenty years it has still not been adequately solved, causing some people to abandon resolution as an effective inference method. The reason often cited is that while resolution is operationally simple (the reason it so appealed to early machine theorem provers), it is very difficult for people to intuitively form or follow a resolution proof. As a consequence, it is also difficult to apply what has been learned about planning and problem solving (based mostly on observation of human problem solving methods) to the resolution inference method.

Knowing this, why was resolution chosen for the QA method? There are several reasons:

1. The first is pragmatic - as has been mentioned, the clausal form was selected partly for reasons other than its applicability to any particular inference method, and resolution is one of the better methods for inference with this form.

2. Special inference methods are used to both speed up the resolution process and as adjunct methods for domains where resolution does not seem appropriate.

3. High level strategies are used to control the proof attempt.

4. Inference for natural language is believed to be shallow, and reasons 2 and 3 should be sufficient to keep the clausal explosion down until a proof is found.

4.5. Enhancement of resolution

Methods to enhance the resolution process work within the process, to speed it up (as compared to methods of control, which operate outside the process). Two enhancements were added: hierarchical resolving and evaluation.

Resolving two clauses is usually done by resolving on literals from each clause which have the same predicate, but opposing signs. For example, [LRRH girl] resolves against with ¯[LRRH girl]. Hierarchical resolving expands this by requiring only that the (signed) predicates be disjoint. For example, [LRRH girl] resolves against [LRRH bird], and ¯[LRRH mammal] resolves against [LRRH girl]. Hierarchical factoring is done in much the same way. Currently, two special inference methods quickly infer the necessary equivalence, disjoint, or subsumptive relationships between predicates. One method infers these relationships for concepts in a type hierarchy (the CH), and another does the same for 'colour' concepts. In the latter case, hierarchical resolving is somewhat of a misnomer, as the colour model used by the special inference method is in fact geometrical rather than hierarchical[6].

Every clause from the original question or generated during the proof attempt goes though an evaluative process. The clause as a whole is evaluated by checking whether it (or its negation) has previously been asserted as a fact. If so, it either signals the end of the

[6] To contradict this contradiction: in the current implementation the colour model is hierarchical, but simply for pragmatic reasons. A geometric model exists, and only awaits an implementation.

proof, or indicates that the clause should not be used in the proof (which is refutational). If the clause cannot be evaluated, each of its literals is tried. Again, a check is done to see if the literal or its negation has been previously asserted as a fact. If so, the literal could either be removed from the clause (if it evaluates to false), or indicate that its embedding clause (a disjunction of literals) is also true. Literal evaluation is also the general method's hook into special inference methods, which could be as simple as a procedural attachment for the 'addition' predicate, or as complex as a complete representational and inferential system for time predicates such as 'before' or 'during'.

4.6. Control of resolution

Control strategies used by the QA method have been divided into two classes: syntactic and semantic. Syntactic strategies operate more of less independently of the underlying meaning of the clauses under consideration, whereas semantic strategies do use those meanings. Of course, at some basic level all methods are just syntactic symbol pushers, but a well defined criterion can be used to separate the two. A method is called syntactic if it can be applied to any and all clauses, regardless of what symbols make up the clause. Semantic methods, on the other hand, examine these symbols, and base their decisions on additional information they have about certain classes of symbols (the implication being that semantic methods have some greater understanding of what these symbols are supposed to mean, aside from their mere presence in the clause). To apply this criterion to the resolution enhancement methods, hierarchical resolution and the special inference methods for evaluating literals (as for time) are semantic methods, whereas the evaluation of clauses or literals based on the normal form is a syntactic method.

4.6.1. Syntactic control

A rather large number of syntactic resolution control strategies have been developed over the years. In varying degrees, all of them have been successful, but even if several are applied at the same time they are not able to contain the clausal explosion. The QA system uses only two of the more successful ones (set of support and preference for least complex resolvents) choosing to rely on semantic methods for most of its control.

To use the set of support strategy, the method simply has to ensure that at least one clause in a resolution pair is either from the original question clauses, or is the resolvent from a previous set of support resolution. This strategy is complete. The intuition behind set of support is that because resolution is a refutational system, resolving two clauses that are known to be true will not add to the refutation. It is essentially backward-chaining for resolution.

To use least complex resolvent preference, the method simply choses to perform resolutions which will lead to resolvents with the least expected complexity. The intuition behind this preference is that, simply (and syntactically) stated, the goal of any resolution proof attempt is to achieve the null clause (a clause with 'zero' complexity), and the best way to do that is to keep the complexity of newly generated clauses as low as possible. The complexity parametrization of a clause is based on the number of literals, constants, variable, and functions that appear in the clause. The number of literals in a clause has the greatest weight in the parametrization, so this method is very similar to a 'fewest number of literals' preference strategy. Note, however, that it is not the same as unit preference, as this method would prefer to resolve two two-literal clauses over resolving a unit clause against a clause with four literals. The expected complexity of a resolvent is calculated by summing the complexities of the parent clause, and subtracting the complexities associated with the literals which are to be resolved on. This preference strategy can be used to a greater extent, and [WOL84]

describes a system in which term complexity plays a major role in the control of inference (although it is left up to the user to determine exactly how to calculate this complexity).

4.6.2. Semantic control

The examination of any uncontrolled resolution proof that has gone astray soon reveals a large number of resolutions where the unification process made some substitution that was not semantically possible, and could never lead to any valid conclusions. For example, it is syntactically possible to resolve A.x ~[x wolf] | [x grey] against ~[LRRH grey] or ~[x frog] | ~[frog grey]. However, we intuitively realize the universal variable 'x' is typed to represent 'wolves', and does not say anything about people or frogs. In this case, performing the first resolution creates the clause ~[LRRH wolf], which is patently useless, but in more complex cases it is not nearly so obvious, and long and fruitless resolution chains are created. Perhaps another way to express this idea is to say that universal variables are not nearly as universal as the syntax of the predicate calculus suggests.

Multi-sorted logics are a formal solution to this problem, but they do require explicit typing of all constants and argument positions, and an extra 'sort' check during unification. The concept hierarchy of the semantic net can also be regarded as a sort structure, and classification as a procedure that automatically sorts constants, variables and, implicitly, argument positions. A universal variable in a clause will almost always appear in the TAS for the concept that represents the type of that variable. For example, ~[x wolf] | [x grey] appears in the TAS for 'wolf', under wolf->colouring. Furthermore, topic access skeletons can be considered as a kind of permanent link between potentially incompatible literals, all of whose arguments are already of roughly the same sort. The QA system takes full advantage of these aspects of the net when it looks for possible resolving actions for a clause. In fact, it uses these structures exclusively, and avoids many of the useless resolving actions noted above.

An additional set of question classifications had to be developed to support the method, for two reasons:

1. Resolution is a refutational system, so in some sense the normal assertive classifications had to be inverted when looking for clause to resolve against. For example, [WOLF wolf] is assertively classified in the CAS (at NET->wolf). When answering a question such as ? E.x [x wolf] (which inverts to ~[x wolf]), we need some way of getting to the same access point.

2. When answering questions, if required knowledge is not found directly at a particular CAS or TAS node, we need to spread out from that node, using the structure of the CH to follow property inheritance paths. Whereas assertorial classifications merely point at a single CAS or TAS node, question classifications must also enumerate possible search paths off that node.

Question classifications are described in more detail Chapter 5.

A further strategy that sortal methods use is to look at the cardinality of the sets of instances for each type, and choose different search paths based on these cardinalities. The QA method takes that concept and applies it to the amount of knowledge available off a particular access point. For example, if trying to answer "Are there any tigers in Edmonton?", and you happen to know of only one particular tiger, which is chained up in your neighbour's yard in Edmonton, then it would probably be best to base the proof on that particular tiger. However, if you were trying answer "Are there people living on Mars?", then it would not be practical to prove that all of the people you know do not live on Mars. Instead, you would probably choose to base your proof on generalities. The QA method makes the same kind of choice when it ranks access points, ranking those which lead to less knowledge above those that lead to a lot of knowledge (all else being equal).

Looking for resolving links at only those points indicated by the classification scheme means that the QA method is very dependent on the accuracy and completeness of that scheme. I would argue that this will always be the case, no matter what kind of inference method is used. This particular method just makes the link that much more explicit, and if the QA method does not seem to 'find' some useful resolving action (or finds too many irrelevant ones), then the problem can be traced back to the classification and organization of the net, which is where it belongs.

The close integration of the QA method with the classification and organizational schemes of the net lead to the obvious conclusion that the implementations of the two could also be integrated. This was in fact done, resulting in very short setup times for questions, and, more pragmatically, shared code and data structures.

4.6.3. Other control strategies

Several other methods were examined to see if they could control resolution in some useful way. I will discuss two of them, and explain why they were not used.

Kowalski, in his connection graph method for resolution [Kow75], introduced the notion of disallowing resolution on two literals if they had previously been resolved on, or if their respective ancestors had been previously resolved on. While this method is logically complete, and seems to reduce the number of resolutions (and therefore clauses), there is a basic problem with the method which is best illustrated by an example.

Given the facts: "All women are persons; all men are persons; GM (grandmother) is a woman, W (woodcutter) is a man; all persons over 70 are old; and GM is over 70", it should be possible to answer the question "Is GM an old person?". The connection graph for this problem appears in Figure 4.1.

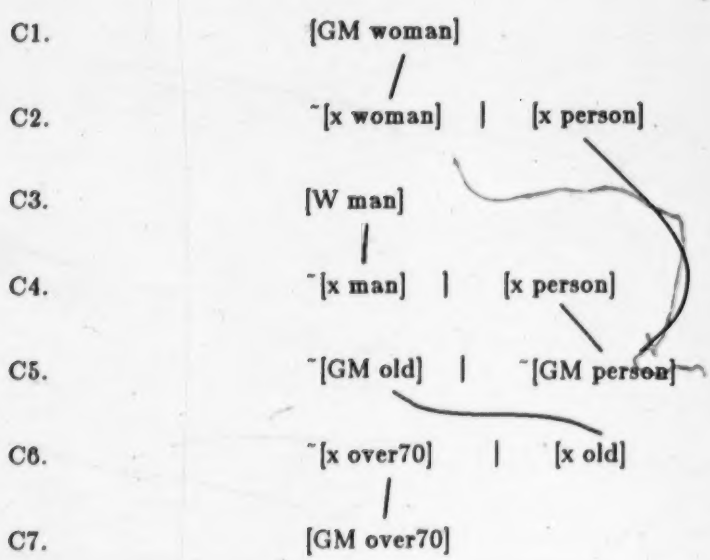


Figure 4.1 Connection graph for the GM problem

Now, the fastest and most obvious solution is to resolve C5 against C2, resolve the result against C1, the result of that against C6, and finally, the result of that against C7. However, if the theorem prover is not quite so capable of perceiving the obvious, it might well resolve C5 against C2, and then, changing course somewhat, resolve C5 (which is still fully connected) against C6, and resolve the result of that against C7. The connection graph after these three resolutions is presented in Figure 4.2.

It should now be possible to resolve C10 against C2, and the result of that against C1, to achieve the null clause in two steps. However, the connection graph method prevents us from doing that, because the first step, resolving C5 against C2, prevented clause 10 from inheriting the connection to C2. In fact, by the method, C2 would be removed entirely, because it is no longer fully connected. The only way to achieve the null clause now is to use three steps, resolving C8 against C1, resolving the result against C6, and the result of that against C7.

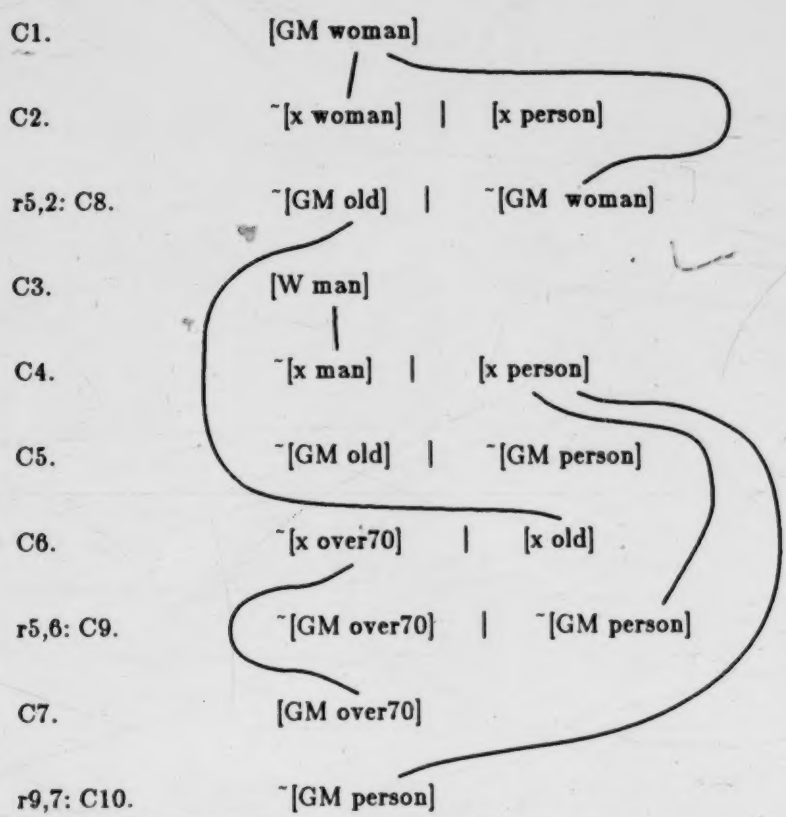

C1. [GM woman]
C2. ¯[x woman] | [x person]
r5,2: C8. ¯[GM old] | ¯[GM woman]
C3. [W man]
C4. ¯[x man] | [x person]
C5. ¯[GM old] | ¯[GM person]
C6. ¯[x over70] | [x old]
r5,6: C9. ¯[GM over70] | ¯[GM person]
C7. [GM over70]
r9,7: C10. ¯[GM person]

Figure 4.2 Graph for the GM problem after three resolutions

In a sense, we are forced to pay for the original mistake by going back to the beginning to do it properly. In this simple case, the price was an additional resolution step, but in more complex cases the price will be higher.

This alone does not mean that the method is unusable, as it might well be possible to prove most control strategies are non-optimal for some problem. In fact, this particular strategy was implemented in an earlier version of the QA system. However, even when used with algorithms that efficiently passed prohibited resolving links on to resolvents, the method incurred a high overhead. After some experience with this strategy (in which the problem just discussed was observed many times), it was decided that the disadvantages of the method

outweighed its advantages.

A second attempt to provide more control was to generate resolution plans, as suggested by [Cha79]. Essentially, the idea is to match all incompatible predicates as possible resolution pairs and to defer unification until you are sure that at least in theory the path could be used to achieve the null clause. Once such a plan is formed, it would then be a matter of finding out whether there existed a set of consistent substitutions which would enable the plan to be carried through. This may well work in cases where the total number of clauses is small, but when this is not the case, the total number of plans that can be generated is enormous - so that heuristics have to be employed to prune these plans. It turned out that we could not find meaningful heuristics without actually looking at the substitutions required by higher levels of the plan, which in turn was equivalent to resolving. For example, if classification is used as the main method of looking for incompatible predicates, substitutions would have to be done before the classification procedure could operate. For these reasons, this kind of resolution planning was not used.

Chapter 5

Implementation

5.1. Description of the algorithm

Very simply, the QA enters the normalized question and its negation in the net, looks for possible resolving links around the access points at which these clauses are classified, and resolves on one of these links. The resolvent is then entered into the net, and the cycle continues until an answer is achieved. The pseudo-code in Figure 5.1 fleshes out this somewhat sketchy description.

5.2. QA control structures

Instead of building elaborate control and data structures for the inference process, the QA makes extensive use of the net's structures. There are only two data structures exclusive to the QA system: an ordered list of access points for clauses in the set of support, and an ordered list of possible resolving actions. Complete control of the algorithm is centered about these two lists, and at the highest level, the only decision the QA system makes is whether to do the best resolving action or to look around the best access point for other resolving actions. Of course, at a lower level, it also has to decide how to order the lists.

5.3. Checkpointing the net

Resolution is a refutational inference method, and the QA system can generate many false clauses during the search for a proof. Central to the QA system's ability to use the net to store these clauses is the ability to checkpoint the net, and to restore it to the checkpointed state after a question is completed. To save information about every little change to a net structure would require a large amount of storage, so several high level deletion primitives

```
Normalize the question and its negation.

Enter these clauses into the net, and add their access
points to the global list of all such points.

REPEAT UNTIL resolving yields the null clause OR
            an entire clause evaluates to false OR
            nothing more can apparently be done OR
            some resource has been exceeded

  IF possible resolving actions have been identified AND
     it appears better to resolve than to look for any
     more THEN

       Try the best resolving action (the best rated
       on the list of all possible resolving actions).

       IF resolving (unification) succeeds THEN

            Enter the new clause into the net and add
            its access points to the global list of
            all such points.

  ELSE
       Look for possible resolving actions at the best
       access point (the best rated in the list of all
       access points). Add possible resolving actions to
       the global list of all such actions.

END REPEAT
```

Figure 5.1 Pseudo-code for the QA algorithm

were developed. After a checkpoint, the system simply keeps track of new concepts, new propositions, and access points for these new propositions. When the checkpointed state is restored, these new concepts, propositions, and access points are deleted. In the latter case, this means that CAS and TAS structures must be contracted back to a minimal skeleton. Algorithms for this process (essentially the inverses of the skeleton building algorithms described in [Cov80]) were also developed. Literals or clauses that existed in the net before it was checkpointed, and which the QA system attempted to evaluate will retain their evaluation after the checkpointed state is restored.

5.4. Concurrent proof and disproof

Concurrent proof and disproof are done by marking every clause in the set of support as belonging to either the proof or disproof set. When searching for possible clauses to resolve against, the QA system only considers clauses from the same set of support, or clauses that are true in general (i.e., asserted as true before the question was asked).

5.5. Entering a clause

The actions performed for a clause originating from the question or its negation and for a clause created by a resolving action are the same:

1. If the clause has been derived before and unsuccessfully evaluated, ignore the clause, and do not enter it again.

2. Try to evaluate the clause, or any of its literals. If the predicate of a literal indicates it is a literal which may be evaluated by a special inference method, the literal is passed to the method for evaluation. If a literal evaluates to true, remove the clause from further consideration. If a literal evaluates to false, remove the literal from the clause and enter the new clause. If a clause evaluates to true, remove it from further consideration. If a clause evaluates to false, an answer has been achieved.

   A negated literal which has been successfully evaluated implies the successful evaluation of the corresponding positive literal. Conversely, the successful evaluation of a positive literal implies the evaluation of its negation.

   Clauses which are asserted or evaluated to be true are marked as such, and the normal form in which clauses are stored makes the evaluation of future instances of the same clause very simple. However, the normal form can not be used to easily find negations of a clause. The exception to this is the original question, where the normalized forms for both the question and its negation are generated anyway.

3. If the clause was not removed by evaluation, it is classified as if it were an assertion, and asserted at the TAS and CAS nodes indicated by the classification. The TAS or CAS will be expanded if necessary.

4. The clause is now classified as a question, which indicates the complete set of TAS or CAS nodes where possible resolving links might be found. The question is not asserted at these classifications - they are merely access points in the net from which the search for possible resolving actions originate. Each questioned access is added to the global list of access points.

   If the clause was classified at the same point twice, a possible factorization is indicated. An attempt is made, and if it is successful the new clause is entered.

The QA system makes one additional check when entering the clauses for the question and its negation, as opposed to entering a resolvent clause. As mentioned above, if any clause evaluates to false, then a proof has been achieved. Normally, a clause which evaluates to true is simply removed from consideration. However, a proof is also achieved if all clauses corresponding to the original question evaluate to true (and the answer is 'yes'), or if all clauses corresponding to the negation evaluate to true (and the answer is 'no').

5.6. Classification of question clauses

Classification is the process of finding which access points (nodes in a TAS or CAS) a clause should be linked to. As was mentioned in the previous section, questioned clauses are classified twice: first as if they were assertions (so that they can later be 'found' by newly generated clauses), and then by a set of special question classifications. Classification of questions is the identification of very specific nodes in the network where possible resolving actions for a clause could be found. The questioned clause is not linked to these nodes - they are merely access points for knowledge which might prove useful in answering the question.

A few remarks about them are in order. Assertorial classification links the entire clause under the appropriate TAS or CAS. However, the classification procedure actually operates on literals, and finds access points specific to one or two literals. Question classification is done to look for resolving actions from these literals, so the distinction is maintained. All sets of TAS or CAS nodes included in access points indicated by question classification are disjoint, so that the search for possible resolving actions is not duplicated. This search space is considered exhaustive, and the QA system will simply say, "don't know" if it can't find any resolving actions off these access points. Unfortunately, it is still possible for the same resolving action to be found by two different access points (if the parent clauses were themselves classified in two different ways), and a further check must be made to ensure that the same resolving action is not done twice.

Six kinds of question classifications are used (Figure 5.2 illustrates the major ones):

1. Topical.

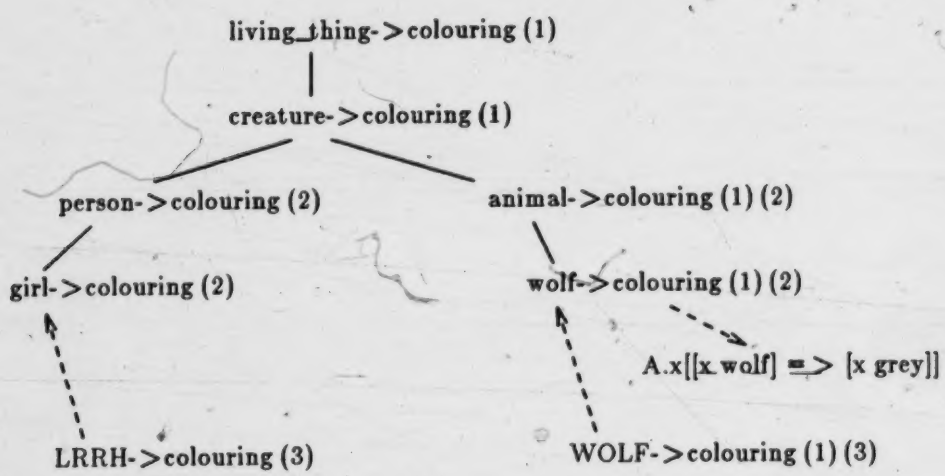
   This is very similar to the topical classification for assertions, but also includes the same topic at super-concepts (generalizations) of the access node. For example, a topical access point of ?[WOLF grey] would include the nodes wolf−>colouring, warm-blooded_quadruped−>colouring, larger_animal−>colouring, animal−>colouring, creature−>colouring, etc. This access point would be used to find the clause ~[x wolf] | [x grey], when answering the question ?[WOLF grey].

2. Major implication.

   Every positive literal is classified under the major implication topic of its predicate. For example, a major implication access point for ?[BASKET eat GM] is in−>majorimp. This access point would find the clause ~[x eat y] | [x living_thing] when answering the question ?[BASKET eat GM].

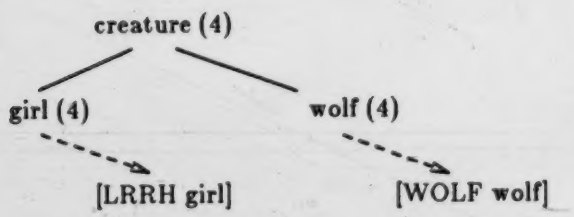
   Most of the work to ensure that major implications are followed should actually be

The 'colouring' topic in a fragment of the CH



(Note: LRRH and WOLF are not part of the CH, but are 'connected' to it via their types.)

A CAS containing the concepts LRRH and WOLF



(1) Topical question access for ? [WOLF grey]

(2) Sub-topical question access for ? A.x [x creature] => [x pink]

(3) Associative topical question access for ? A.x [x creature] => [x pink]

(4) Associative question access for ? E.x [x creature]

Figure 5.2 Question classifications

done during the initial assimilation of facts. Some variant of the QA system would be run on each new fact as it is entered, asking at least if it contradicts previous knowledge. For example, given the fact that A is in B, the system would use what it already knows about the sizes of A and B to make sure that it is physically possible for A to be in B. If it couldn't prove that it is impossible, the system would add the additional fact that A is smaller than B. Currently, no consistency checking is done for new facts, and the system uses major implications only when directly answering a question.

3. Exclusion.

Every positive literal is classified under the exclusion topic of its predicate. For example, an exclusion access point for ?[[LRRH thin] & [LRRH fat]] would be fat−>exclusion. This access point would find the clause ~[x thin] | ~[x fat] when answering the question ?[[LRRH thin] & [LRRH fat]].

4. Associative.

This classification is used for literals which consist of a variable and a type predicate which did not get classified topically. For example, an associative access point for ?A.x [x creature] could include the nodes NET−>creature, NET−>animal, NET−>person, etc. This access point would find the clause [LRRH girl], when answering the question ?E.x [x creature].

5. Sub-topical.

This classification includes the same topic at all sub-concepts (specializations) of a concept−>topic pair. It does not include the original concept−>topic pair, and is restricted to general concepts (i.e., it does not extend to instances). For example, a sub-topical access point for ?A.x [[x creature] => [x pink]] includes the nodes wolf−>colouring, person−>colouring, bear−>colouring, elephant−>colouring, etc.

This access point would be used to find the clause ~[x wolf] | [x grey], when answering the question ?A.x [[x creature] => [x pink]] (i.e., to find a generic example of a creature that is not pink).

6. Associative-topical.

This classification is similar to sub-topical, except that it only applies to instances. For example, an associative-topical access point for ?E.x E.y [[x creature] & [y hood] & [x wear y]] could include the nodes LRRH->apparel, WOLF->apparel, etc. This access point would find the clause E.x [LRRH wear x] (where x is also typed as [x hood]) when answering the question ?E.x E.y [[x creature] & [y hood] & [x wear y]] (i.e., it finds an instance of a creature wearing a hood).

There is one further question classification, but it can be considered a combination of sub-topical and associative-topical classifications. It occurs for a literal consisting of a variable and a non-type predicate which could not get classified topically. For example, a sub/associative-topical access point for ?A.x [x red] includes the colouring topic for all sub-concepts of 'top-of-hier' (the node at the top of the CH) and the colouring topic for all known concepts in the CAS. This kind of access point would find the clauses E.x [[x hood] & [x red]] when answering the question ?E.x [x red].

5.7. Ranking access points

All access points are initially assumed to be equally probable as points at which a successful resolving action will be found. Given this assumption, access points can still be ranked so that points with the least amount of knowledge associated with them are done first (the implementation 'least knowledge first' ordering for search paths). The heuristic used to estimate the amount of knowledge is the total number of clauses that are classified at or

about the access point. In some cases, this is an exact amount, maintained by the net, and in other cases guesswork is involved. In all cases, clauses from the opposite set of support (which should not be considered at all) will throw the count off. Clause counts for the various question classifications are derived as follows:

1. Topical, major implication, exclusion access points.

   The number of clauses for each of these access points is maintained at the corresponding access skeleton node, and is directly available. Topical clause counts should also include the counts at the same topic of concept generalizations ('super-topical' access points), but this is currently not done.

2. Associative access points.

   The number of clauses stored at or below the corresponding CAS node is again automatically maintained by the system during assertion.

3. Sub-topical access points.

   The clause count for this kind of access point can only be estimated. First, the number of concepts in the CH below the concept for which the access occurs is calculated (a simple matter using a numbering scheme on the CH). Then, this number is multiplied by the average number of clauses classified under at that topic (i.e., the total number of times the classification 'any_concept->topic' occurs, divided by the total number of unique concepts in 'any_concept'). This average is maintained during assertion.

4. Associative-topical access points.

   The clause count for this access point is also a guess. First, the number of concepts is derived by looking at the number of clauses which appear in the CAS at or below the current node, which is assumed to be equivalent to the number of clauses at or below this point. This is then multiplied by the topic average (see sub-topical).

Finally, the complexity of the clause which indicated the access point is taken into consideration, so that access points for simpler clauses are done first (the current assumption is that every literal less is worth looking at about 10 more clauses; for example, the QA system would expand an access for a unit clause with 19 prospective clauses before expanding an access for a two literal clause with 10 clauses).

5.8. Ranking possible resolving actions

Possible resolving actions are ranked according to the expected complexity of the resolvent, which is in turn based on the complexity of the two parent clauses. Resolvents with the least expected complexity are tried first.

Clause complexity in general is calculated using the number of literals (most weight), the number of arguments per literal, argument types (variable, constants, or function) and argument complexity (number of arguments to a function).

5.9. Unification and resolving

At least one linear (in time) unification algorithm appears in the literature [Bax73], but the constant (setup) time associated with this algorithm seems somewhat excessive, so a near-linear algorithm was used instead. The algorithm needs very little setup time, and is modeled on the union-find algorithm described in [AHU74].

5.10. Implementation details

The QA system is expected to be somewhat of an experimental tool, and to aid its use as such, a 'user friendly' front end was written for it. Aside from accepting propositions and questions, the system allows a user to examine the concept hierarchy, concept access skeletons, the topic hierarchy and topic access skeletons, to print various details about concepts and propositions, to set parameters for question answering (such as the number of iterations to perform before giving up), and to trace the progress of a question answering attempt. The trace mechanism can be adjusted to print only the barest minimum or to dump absolutely everything in excruciating detail. Sample output is provided in Appendix A.

The QA system was written in Berkeley PASCAL, which runs under UNIX 4.2 BSD on a Digital Equipment Corporation VAX computer.

Chapter 6

Conclusions

The major task of this thesis was to design a system which could be used to support inference needed for natural language understanding, using the representational and organizational structures of the semantic net. Such a system was in fact designed, and implemented as a PASCAL program. It is able to answer a test set of 40 questions in about 15 seconds of CPU time on a VAX 780[7] (Appendix A lists these questions). Approximately 200 clauses were used in this test. In another test, doubling the size of the knowledge base (to about 400 clauses) had no effect on the time needed to answer these same 40 questions.

Control of the resolution-based inference method relies very much on semantic information provided by the organizational structure of the net, with additional control provided by the syntactic methods of set of support and preference for least complex resolvents. The resolution step is further enhanced by special inference methods for hierarchical resolving and literal evaluation. The inference method is logically sound, and although resolution itself is logically complete, additional constraints imposed by the implementation most likely ensure that this method is not complete.

The successful implementation of the method indicates that the original design of the net's organizational structure is sufficient to provide quick access to the relevant knowledge needed for low-level inference for natural language understanding. Furthermore, the method also indicates that the organizational structure can be used to directly guide and constrain deduction with that knowledge. The processes of access and inference become virtually inseparable, which leads to two other distinguishing characteristics of the method: (1) a clean

[7] Note that a VAX 780 is only a mini-computer, and 15 seconds of CPU time on this machine is roughly equivalent to about 1 or 2 seconds on the MTS Amdahl machine.

and simple implementation of the method was done within the implementation of the net (as opposed to a separate inference 'engine'); and (2) the deductive ability (or inability) of the method can be traced back (to a large extent) to the net's organizational ability, where it is much more visible than if buried in code, and hence easier to understand, explain, and fix.

6.1. Future research

The general representational scheme of the net needs to be extended to allow certain kinds of knowledge to be expressed more easily (as discussed in Section 4.3) and corresponding extensions to the inference method will also have to be made. The net is already capable of representing and organizing modal propositions such as "John believes that ...", but to properly reason about further beliefs that John might have the inference method will have to incorporate some of its own beliefs about John's reasoning processes (or lack thereof).

One of the main purposes of the topical hierarchy (and associated skeletons) is to aid query processing, by quickly gathering all topically relevant knowledge about an individual concept. Thus, the process that answers queries such as "What do wolves look like?", gathers clauses from all TAS nodes at or below wolf−>appearance, including sub-topics such as wolf−>colour. If the question were changed to "What does the WOLF look like?", some rudimentary inference is needed to specialize the knowledge about wolves in general to the WOLF in particular. The QA method does not currently process queries of this sort, although it is certainly capable of making the required specialization inferences. Query processing is discussed at some depth in [Cov80].

The QA method does use TAS nodes to search for incompatible literals. However, it assumes incompatible literals will always be found at the same TAS node, and it doesn't expand the search up or down the TAS. This is not an unreasonable assumption, given the current state of the classification procedure, and the lack of special inference methods to

determine incompatibility other than by opposite signs on the same predicate. However, should these methods become available then the search for incompatible literals should be broadened accordingly.

One immediate change to the classification procedure would be to mark literals it uses to classify asserted clauses. Instead of pointing directly at a clause, a TAS node would point at both the clause and the literals that caused it be be classified at that node. Then, instead of searching whole clauses for incompatible literals, as it does now, the QA system could get to the 'right' literal immediately. This would save some time, but more importantly, it would prevent the system from making 'non-semantic' resolutions. This is especially a problem when using longer clauses in which most of the literals refer to the same topic.

The current implementation of the QA system only includes a special inference method for types, and a rudimentary method for colours. A special inference method for time is in the process of being added, but other methods should be considered, such as for parts, or relationships between physical objects. Special domain independent inference methods could also be used, such as a method for transitivity. Some work will have to be done to successfully incorporate evaluative special inference methods into a refutational proof procedure.

The single taxonomic structure of the concept hierarchy is not sufficient to organize all the generalization/specialization knowledge that people appear to maintain. For example, a taxonomy for occupations would be useful ('professional' vs. 'non-professional', 'engineer' vs. 'teacher', etc.). Ultimately, a (possibly adaptive) process will be needed to easily apply any generalization to specializations or instances (i.e., property inheritance), but in the short term the net could benefit from the addition of a few more 'hand-built' hierarchies.

Limited forward inferencing could be implemented by keeping 'interesting' clauses formed during the assimilative inference needed for asserted clauses. However, work needs

to be done on methods which can accurately decide between interesting clauses and what should be discarded (the application of a major implication could be one criterion used by such a method).

One promising method of improving the control of a proof attempt is to give concepts (or topics, or propositions, or even hierarchies) salience ratings. Given a choice between two deductive steps, the QA system would then choose the step which involved concepts which overall had a higher salience. Salience could be derived from the discourse, from the initial form of a question, or by the inference method itself.

Currently, the QA system makes evaluation results available to both the proof and disproof attempts, but it seems desirable to share more of the inference between the two. One method would be to repeat every backward inference that used literals from asserted clauses as a forward inference. However, the value of this method appears limited, for two reasons: (1) a great deal of knowledge is expressed as implications, which are not logically strong enough to prove their inverses (e.g., knowing that all baseball players chew gum is sufficient to prove that you are not a baseball player if you don't chew gum, but it is not sufficient to prove that you are a baseball player if you do chew gum); and (2) a lot of the inference that is currently duplicated is that of establishing that generalized knowledge is applicable to some individual, and this inference is done as a single hierarchical resolution step anyway. It is possible that resolution is not the right inference method for such a system, or that forward inferencing during assimilation would be sufficient.

It should be possible to derive better heuristics for some of the clause counts used by the system, and for the parametrization of clause complexity. Historical information and statistical methods could probably come up with better weighting factors than are used now.

The adequacy of this method as a base for higher level inference methods can only be

verified by the development of those methods. One of the first steps would be to expand the current 'yes/no' answer into a more informative answer. There already exist methods to extract information from a resolution proof tree (see [LuN71]), but these have not been implemented. A further step, as discussed in [Bai86], would be to generate natural language responses from the proof or disproof clause set.

References

[AHU74] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[Bai86] A. J. Bailes, *Response Generation*, M.Sc. Thesis, University of Alberta, 1986.

[Bax73] L. D. Baxter, *An Efficient Unification Algorithm*, University of Waterloo, Waterloo, Ontario, 1973.

[Cha79] C. L. Chang, Resolution plans in theorem proving, *IJCAI-6*, Tokyo, Japan, 1979, 143-148.

[Coh83] A. G. Cohn, Improving the expressiveness of many sorted logic, *AAAI-83*, Washington, D.C., 1983, 84-87.

[Coh85] A. G. Cohn, On the solution of Schubert's Steamroller in many sorted logic, *IJCAI-9*, Los Angeles, 1985, 491-501.

[CoS80] A. R. Covington and L. K. Schubert, Organization of modally embedded propositions and of dependent concepts, *Third National Conference of the CSCSI/SCEIO*, Victoria, BC, May 14-16, 1980, 87-94.

[Cov80] A. R. Covington, *Organization of Modally Embedded Propositions and of Dependent Concepts*, M.Sc. Thesis, University of Alberta, 1980.

[FaH85] R. Fagin and J. Y. Halpern, Belief, awareness, and limited reasoning: preliminary report, *IJCAI-9*, Los Angeles, 1985, 491-501.

[FiH77] R. Fikes and G. Hendrix, A network-based knowledge representation and its natural deduction system, *IJCAI-5*, Cambridge, Massachusetts, 1977, 235-246.

[Kol81] J. L. Kolodner, Organization and retrieval in a conceptual memory for events or CON54, Where are you?, *IJCAI-7*, Vancouver, 1981, 227-233.

[Kow75] R. Kowalski, A proof procedure using connection graphs, *Journal of the ACM*, 1975, 572-595.

[LDJ83] W. G. Lehnert, M. G. Dyer, P. N. Johnson, C. J. Yang and S. Harley, BORIS - An experiment in in-depth understanding of narratives, *Artificial Intelligence 20*, (1983), 15-62, MIT Press.

[LuN71] D. Luckham and N. J. Nilsson, Extracting information from resolution proof trees, *Artificial Intelligence 2*, (1971), 27-54.

[McM77] J. R. McSkimin and J. Minker, The use of a semantic network in a deductive question-answering system, *IJCAI-5*, Cambridge, Massachusetts, 1977, 50-58.

[Nil80] N. J. Nilsson, *Principles of Artificial Intelligence*, Tioga, Palo Alto, 1980.

[Pat85] P. F. Patel-Schneider, A decidable first-order logic for knowledge representation, *IJCAI-9*, Los Angeles, 1985, 455-458.

[Rob65] J. Robinson, A machine-oriented logic based on the resolution principle, *Journal of the ACM*, 1965, 23-41.

[Rus85] S. Russell, The Compleat Guide to MRS, Stanford Knowledge Systems Laboratory Report No. KSL-85-12, Computer Science Department, Stanford University, Stanford, California, 1985.

[Sch76] L. K. Schubert, Extending the expressive power of semantic networks, *Artificial Intelligence 7*, (1976), 163-198.

[SGC79] L. K. Schubert, R. G. Goebel and N. J. Cercone, The structure and organization of a semantic net for comprehension and inference, in *Associative Networks*, N. V. Findler (ed.), Academic Press, 1979, 121-175.

[SPT83] L. K. Schubert, M. A. Papalaskaris and J. Taugher, Determining type, part, color, and time relationships, *Computer (USA) 16*, 10 (1983), 53-60.

[Sha79] S. C. Shapiro, The SNePS semantic network processing system, in *Associative Networks*, N. V. Findler (ed.), Academic Press, 1979, 179-203.

[Wal83] C. Walther, A many-sorted calculus based on resolution and paramodulation, *IJCAI-8*, Karlsruhe, West Germany, 1983, 882-891.

[Wal85] C. Walther, A mechanical solution of Schubert's Steamroller by many-sorted resolution, *Artificial Intelligence 26*, (1985), 217-224.

[WOL84] L. Wos, R. Overbeek, E. Lusk and J. Boyle, in *Automated Reasoning, Introduction and Applications*, Englewood Cliffs, New Jersey, 1984.

Appendix A

Sample Output

A.1. Questions and timing

The QA system answered the following 40 questions in a total of about 15 seconds of CPU time on a VAX 780. CPU times include the time needed to parse and normalize the question, so the time to actually answer these questions is slighty less. The complexity of these questions varies, and it takes less time to answer simpler questions. For example, the first seven questions were answered in a total of about 1.5 seconds. The knowledge base used to answer these questions appears in the next section.

1. ? [WOLF grey]
2. ? [BASKET eat GM]
3. ? [[LRRH thin] & [LRRH fat]]
4. ? E.x [x creature]
5. ? A.x [[x creature] => [x pink]]
6. ? E.x E.y [[x creature] & [y hood] & [x wear y]]
7. ? E.x [x red]
8. ? [LRRH rock]
9. ? E.x [[x rock] & [x person] & [x live-in COTTAGE]]
10. ? A.x [[x rock] => [x person]]
11. ? E.x [[x basket] & [WOLF live-in x]]
12. ? [BASKET live-in COTTAGE]
13. ? E.x [LRRH wear x]
14. ? E.x [[x red] & [LRRH wear x]]
15. ? E.x [[x hood] & [LRRH wear x]]
16. ? A.x [[x wolf] => [x grey]]
17. ? A.x [[x cape] => [x blue]]
18. ? E.x [x grey]
19. ? E.x [WOLF eat x]
20. ? E.x E.y [[x creature] & [y creature] & [x eat y]]
21. ? E.x [[x rock] & [x eat GM]]
22. ? [COTTAGE eat GM]
23. ? E.x [[GM wear x] & [WOLF wear x]]
24. ? E.x E.y [[x kill y] & [y creature]]
25. ? E.x E.y [[x kill y] & [y grey]]
26. ? E.x E.y [[x person] & [y wolf] & [x kill y]]
27. ? E.x E.y [[x creature] & [x kill y] & [y grey]]
28. ? [WOLF kill BASKET]
29. ? E.x [[x cake] & [LRRH eat x]]
30. ? [WOLF like-to-eat LRRH]

```
31. ? A.x [[x cake] => [LRRH like-to-eat x]]
32. ? [LRRH like-to-eat BASKET]
33. ? E.x [[x in BASKET] & [LRRH like-to-eat x]]
34. ? E.x [[x in BASKET] & ~[LRRH like-to-eat x]]
35. ? A.x [[x in BASKET] => [LRRH like-to-eat x]]
36. ? E.x [[x teeth-of WOLF] & [x sharp]]
37. ? E.x [[x teeth-of GM] & [x sharp]]
38. ? E.x [[x ear-of WOLF] & [x big]]
39. ? E.x [[x ear-of GM] & [x big]]
40. ? E.x [[x eye-of WOLF] & [x big]]
```

A.2. The LRRH knowledge base

{ This set of propositions asserts the concept hierarchy }

```
A.x [[x living_thing]      => [x physical_object]]
  A.x [[x plant]           => [x living_thing]]
    A.x [[x leafy_plant]   => [x plant]]
      A.x [[x tough-stemmed_plant] => [x leafy_plant]]
        A.x [[x tree]      => [x tough-stemmed_plant]]

  A.x [[x creature]        => [x living_thing]]
    A.x [[x person]        => [x creature]]
      A.x [[x adult]       => [x person]]
        A.x [[x man]       => [x adult]]
        A.x [[x woman]     => [x adult]]

      A.x [[x minor]       => [x person]]
        A.x [[x child]     => [x minor]]
          A.x [[x girl]    => [x child]]


    A.x [[x animal]        => [x creature]]
      A.x [[x larger_animal]   => [x animal]]
        A.x [[x warm-blooded_quadrudped] => [x larger_animal]]
        A.x [[x wolf]      => [x warm-blooded_quadruped]]

A.x [[x non-living_thing]  => [x physical_object]]
  A.x [[x inanimate_natural_object] => [x non-living_thing]]
    A.x [[x forest]        => [x inanimate_natural_object]]
  A.x [[x artifact]        => [x non-living_thing]]
    A.x [[x device]        => [x artifact]]
      A.x [[x hand_tool]   => [x device]]
        A.x [[x axe]       => [x hand_tool]]
      A.x [[x cutlery]     => [x device]]
        A.x [[x butter-knife]  => [x cutlery]]
        A.x [[x fork]      => [x cutlery]]
        A.x [[x spoon]     => [x cutlery]]
    A.x [[x furniture]     => [x artifact]]
```

```
    A.x [[x bed]                  => [x furniture]]
  A.x [[x container]              => [x artifact]]
    A.x [[x basket]               => [x container]]
  A.x [[x architectural_object]   => [x artifact]]
    A.x [[x building]             => [x architectural_object]]
      A.x [[x cottage]            => [x building]]
    A.x [[x shelter]              => [x architectural_object]]
      A.x [[x den]                => [x shelter]]
  A.x [[x apparel]                => [x artifact]]
    A.x [[x nightgown]            => [x apparel]]
    A.x [[x nightcap]             => [x apparel]]
    A.x [[x cape]                 => [x apparel]]
    A.x [[x hood]                 => [x apparel]]
  A.x [[x cover]                  => [x artifact]]
    A.x [[x tablecloth]           => [x cover]]
A.x [[x food]                     => [x non-living_thing]]
  A.x [[x person_food]            => [x food]]
    A.x [[x baked_food]           => [x person_food]]
      A.x [[x pie]                => [x baked_food]]
      A.x [[x cake]               => [x baked_food]]
      A.x [[x cookie]             => [x baked_food]]
      A.x [[x tart]               => [x baked_food]]
    A.x [[x natural_food]         => [x person_food]]
      A.x [[x honey]              => [x natural_food]]
      A.x [[x apple]              => [x natural_food]]
A.x [[x non-living_creature]      => [x non-living_thing]]


{ major implications }

major on
A.x A.y [[x eat y]        => [x living_thing]]
A.x A.y [[x in y]         => [x smaller-than y]]
A.x A.y [[x live-in y]    => [x smaller-than y]]
A.x A.y [[x live-in y]    => [x living_thing]]
A.x A.y [[x kill y]       => [y living_thing]]
major off

{ general knowledge }

A.x [[x fat] => ~[x thin]]

A.x A.y [[[x person] & [x eat y]] => [y person_food]]

A.x [[x wolf] => [x grey]]

A.x E.y E.z
  [[[x person] | [x larger_animal]] =>
              [[y head-of x] & [z body-of x]]]

A.x E.y [[x wolf] => [y ear-of x]]
A.x A.y [[[x wolf] & [y ear-of x]] => [[y big] & [y hairy]]]
```

```
A.x E.y [[x person] => [y ear-of x]]
A.x A.y [[[x person] & [y ear-of x]] => [~[y big] & ~[y hairy]]]

A.x E.y [[x wolf] => [y teeth-of x]]
A.x A.y [[[x wolf] & [y teeth-of x]] => [[y sharp] & [y big] &
                                         [y pointy] & [y white]]]

A.x E.y [[x person] => [y teeth-of x]]
A.x A.y [[[x person] & [y teeth-of x]] => [~[y sharp] & ~[y big] &
                                           ~[y pointy] & [y white]]]

A.x E.y [[x wolf] => [y eye-of x]]
A.x A.y [[[x wolf] & [y eye-of x]] => [[y big]]]

A.x E.y [[x person] => [y eye-of x]]
A.x A.y [[[x person] & [y eye-of x]] => [~[y big]]]

A.x A.y A.z [[[x hood] & [y wear x] & [z head-of y]] => [x on z]]

A.x [[x person] => A.y [[y person_food] => [x like-to-eat y]]]
A.x [[x person] => A.y [~[y person_food] => ~[x like-to-eat y]]]
A.x [[x wolf] => A.y [[y person] => [x like-to-eat y]]]
E.x [x rock]


{ Little Red Riding Hood knowledge }

[LRRH girl]
[GM woman]
[WOLF wolf]
[WOODCUTTER man]
[BASKET basket]
[COTTAGE cottage]
[BED bed]


{ These propositions are used to simulate a special
  inference method for physical size relationships }


~[LRRH smaller-than BASKET]     [BASKET smaller-than LRRH]
 [LRRH smaller-than COTTAGE]    ~[COTTAGE smaller-than LRRH]
 [LRRH smaller-than BED]        ~[BED smaller-than LRRH]

~[GM smaller-than BASKET]     [BASKET smaller-than GM]
 [GM smaller-than COTTAGE]    ~[COTTAGE smaller-than GM]
 [GM smaller-than BED]        ~[BED smaller-than GM]

~[WOODCUTTER smaller-than BASKET]    [BASKET smaller-than WOODCUTTER]
 [WOODCUTTER smaller-than COTTAGE]   ~[COTTAGE smaller-than WOODCUTTER]
 [WOODCUTTER smaller-than BED]       ~[BED smaller-than WOODCUTTER]

~[WOLF smaller-than BASKET]      [BASKET smaller-than WOLF]
```

[WOLF smaller-than COTTAGE] ~[COTTAGE smaller-than WOLF]
[WOLF smaller-than BED] ~[BED smaller-than WOLF]

[BASKET smaller-than BED] ~[BED smaller-than BASKET]
[BASKET smaller-than COTTAGE] ~[COTTAGE smaller-than BASKET]
[BED smaller-than COTTAGE] ~[COTTAGE smaller-than BED]

A.x [[x basket] => [x smaller-than WOLF]]
A.x [[x basket] => ~[WOLF smaller-than x]]

E.x E.y [[x cape] & [x red] & [LRRH wear x] &
[y hood] & [y red] & [LRRH wear y] &
[y attached-to x]]

[GM grandmother-of LRRH]

[GM live-in COTTAGE]

[BED in COTTAGE]

[LRRH carry BASKET]

E.w E.x E.y E.z [[w honey] & [w in BASKET] &
[x cake] & [x in BASKET] &
[y cookie] & [y in BASKET] &
[z pie] & [z in BASKET]]

E.x E.y E.z [[x tablecloth] & [x in BASKET] &
[y butter-knife] & [y in BASKET]]

[LRRH meet WOLF]

[WOLF in COTTAGE]

[WOLF eat GM]

E.x E.y [[x nightgown] & [GM wear x] & [WOLF wear x] &
[y nightcap] & [GM wear y] & [WOLF wear y]]

[WOLF on BED]

[LRRH in COTTAGE]

[WOODCUTTER in COTTAGE]

[WOODCUTTER kill WOLF]

A.x [[[x person_food] & [x in BASKET]] => [[GM eat x] &
[LRRH eat x] &
[WOODCUTTER eat x]]]

A.3. Access skeltons for the knowledge base

The following are printouts of the concept access skeleton for the LRRH story and topic access skeletons for LRRH, the WOLF and wolves (i.e., the generic wolf).

```
**
print skeleton

Concept access skeleton for MAIN net

 physical_object    18 proposition(s):
  creature    4 proposition(s):
   wolf    1 proposition(s):
          [WOLF wolf] (372)
   person    3 proposition(s):
    adult    2 proposition(s):
     man    1 proposition(s):
          [WOODCUTTER man] (374)
     woman    1 proposition(s):
          [GM woman] (370)
    girl    1 proposition(s):
          [LRRH girl] (368)
  non-living_thing    14 proposition(s):
   person_food    4 proposition(s):
    baked_food    3 proposition(s):
     pie    1 proposition(s):
          [ident007 pie] (449)
     cookie    1 proposition(s):
          [ident006 cookie] (452)
     cake    1 proposition(s):
          [ident005 cake] (455)
    honey    1 proposition(s):
          [ident004 honey] (458)
   artifact    9 proposition(s):
    butter-knife    1 proposition(s):
          [ident009 butter-knife] (461)
    tablecloth    1 proposition(s):
          [ident008 tablecloth] (464)
    apparel    4 proposition(s):
     nightcap    1 proposition(s):
          [ident012 nightcap] (472)
     nightgown    1 proposition(s):
          [ident011 nightgown] (476)
     hood    1 proposition(s):
          [ident003 hood] (438)
     cape    1 proposition(s):
          [ident002 cape] (441)
    bed    1 proposition(s):
          [BED bed] (380)
```

```
    cottage    1 proposition(s):
            [COTTAGE cottage] (378)
    basket    1 proposition(s):
            [BASKET basket] (376)
  rock    1 proposition(s):
            [ident001 rock] (366)


**
print skeleton LRRH

Topic access skeleton for concept LRRH

 tp.top    12 proposition(s):
  tp.top-of-hier    11 proposition(s):
   tp.feeding    1 proposition(s):
            [~[netv001 person_food] | [LRRH eat netv001] |
                                    ~[netv001 in BASKET]] (488)
   tp.apparel    2 proposition(s):
            [LRRH wear ident003] (436)
            [LRRH wear ident002] (439)
   tp.static_relationsh    8 proposition(s):
    tp.kinship    1 proposition(s):
            [GM grandmother-of LRRH] (442)
    tp.physical_relation    7 proposition(s):
     tp.location    1 proposition(s):
            [LRRH in COTTAGE] (478)
     tp.size_relationship    6 proposition(s):
      tp.size_from    3 proposition(s):
            ~[BED smaller-than LRRH] (389)
            ~[COTTAGE smaller-than LRRH] (386)
            [BASKET smaller-than LRRH] (383)
      tp.size_to    3 proposition(s):
            [LRRH smaller-than BED] (387)
            [LRRH smaller-than COTTAGE] (384)
            ~[LRRH smaller-than BASKET] (382)
  tp.gen    1 proposition(s):
            [LRRH girl] (368)
**


print skeleton WOLF

Topic access skeleton for concept WOLF

 tp.top    15 proposition(s):
  tp.top-of-hier    14 proposition(s):
   tp.existance    1 proposition(s):
            [WOODCUTTER kill WOLF] (480)
   tp.apparel    2 proposition(s):
            [WOLF wear ident012] (470)
            [WOLF wear ident011] (474)
   tp.feeding    1 proposition(s):
```

```
            [WOLF eat GM] (468)
      tp.physical_relation    10 proposition(s):
        tp.location    2 proposition(s):
            [WOLF on BED] (477)
            [WOLF in COTTAGE] (467)
        tp.size_relationship    8 proposition(s):
          tp.size_from    4 proposition(s):
            [~[netv001 basket] | [netv001 smaller-than WOLF]] (429)
            ~[BED smaller-than WOLF] (416)
            ~[COTTAGE smaller-than WOLF] (413)
            [BASKET smaller-than WOLF] (410)
          tp.size_to    4 proposition(s):
            [~[netv001 basket] | ~[WOLF smaller-than netv001]] (432)
            [WOLF smaller-than BED] (414)
            [WOLF smaller-than COTTAGE] (411)
            ~[WOLF smaller-than BASKET] (409)
    tp.gen    1 proposition(s):
            [WOLF wolf] (372)
**


print skeleton wolf

Topic access skeleton for concept wolf

  tp.top-of-hier    12 proposition(s):
    tp.feeding_likes    1 proposition(s):
      [~[netv001 person] | ~[netv002 wolf] | [netv002 like-to-eat netv001]] (364)
    tp.has_part    10 proposition(s):
      [~[netv001 wolf] | [netv002 big] | ~[netv002 eye-of netv001]] (341)
      [~[netv001 wolf] | [(func007 netv001) eye-of netv001]] (337)
      [~[netv001 wolf] | [netv002 white] | ~[netv002 teeth-of netv001]] (319)
      [~[netv001 wolf] | [netv002 pointy] | ~[netv002 teeth-of netv001]] (321)
      [~[netv001 wolf] | [netv002 big] | ~[netv002 teeth-of netv001]] (322)
      [~[netv001 wolf] | [netv002 sharp] | ~[netv002 teeth-of netv001]] (324)
      [~[netv001 wolf] | [(func005 netv001) teeth-of netv001]] (314)
      [~[netv001 wolf] | [netv002 hairy] | ~[netv002 ear-of netv001]] (301)
      [~[netv001 wolf] | [netv002 big] | ~[netv002 ear-of netv001]] (303)
      [~[netv001 wolf] | [(func003 netv001) ear-of netv001]] (296)
    tp.colouring    1 proposition(s):
      [~[netv001 wolf] | [netv001 grey]] (281)
```

A.4. Tracing the QA system

The trace feature of the QA system instructs the system to print the details of its search for a proof or disproof. For example, here is an extensive trace for the question "Is the WOLF grey?":

```
?[WOLF grey]

  Entering disproof clauses ...
  Entering clause: [WOLF grey] (at depth 1)
  Evaluated [WOLF grey] in [WOLF grey] to unknown
  The clause [WOLF grey] evaluated to unknown

  Entering proof clauses ...
  Entering clause: ~[WOLF grey] (at depth 1)
  Evaluated ~[WOLF grey] in ~[WOLF grey] to unknown
  The clause ~[WOLF grey] evaluated to unknown

Disproof ...
  Unsuccessful exclusion access at 'grey->tp.exclusion'
  Unsuccessful major implication access at 'grey->tp.majorimp'
    Compatibility test:  'grey' is not incompatible with '~wolf'
    Compatibility test:  'grey' is not incompatible with 'grey'
    Unsuccessful super-topical access at 'wolf->tp.colouring'
    Unsuccessful super-topical access at 'warm-blooded_quadrup->tp.colouring'
    Unsuccessful super-topical access at 'larger_animal->tp.colouring'
    Unsuccessful super-topical access at 'animal->tp.colouring'
    Unsuccessful super-topical access at 'creature->tp.colouring'
    Unsuccessful super-topical access at 'living_thing->tp.colouring'
    Unsuccessful super-topical access at 'physical_object->tp.colouring'
    Unsuccessful super-topical access at 'tp.spec->tp.colouring'
    Unsuccessful super-topical access at 'tp.top->tp.colouring'
  Unsuccessful topical access at 'WOLF->tp.colouring'

Proof ...
    Compatibility test:  '~grey' is not incompatible with '~wolf'
    Compatibility test:  '~grey' is incompatible with 'grey'

  Formed a resolving link between
     ~[WOLF grey] (494)
     [~[netv001 wolf] | [netv001 grey]] (281)
    Successful super-topical access at 'wolf->tp.colouring'
    Unsuccessful super-topical access at 'warm-blooded_quadrup->tp.colouring'
    Unsuccessful super-topical access at 'larger_animal->tp.colouring'
    Unsuccessful super-topical access at 'animal->tp.colouring'
    Unsuccessful super-topical access at 'creature->tp.colouring'
    Unsuccessful super-topical access at 'living_thing->tp.colouring'
    Unsuccessful super-topical access at 'physical_object->tp.colouring'
```

```
    Unsuccessful super-topical access at 'tp.spec->tp.colouring'
    Unsuccessful super-topical access at 'tp.top->tp.colouring'
  Successful topical access at 'WOLF->tp.colouring'
  Resolving ~[WOLF grey] (494)
    against [~[netv001 wolf] | [netv001 grey]] (281)
    Resolving literal ~[WOLF grey]  against  [netv001 grey]
  Unifying substitution:
    WOLF/netv001-2
  Result: ~[WOLF wolf] (495)

  Entering this clause ...  (at depth 2)
  Evaluated ~[WOLF wolf] in ~[WOLF wolf] to false
  The clause ~[WOLF wolf] evaluated to false
  The answer is yes.
```

The trace can be set to record only 'successful actions'. For example, here is a trace of the successful actions for the question, "Is there something in the BASKET that LRRH likes to eat?"

```
?E.x [[x in BASKET] & [LRRH like-to-eat x]]

  Cleaning up after previous question.

  Entering disproof clauses ...
  Entering clause: [ident013 in BASKET] (at depth 1)

  Entering clause: [LRRH like-to-eat ident013] (at depth 1)

  Entering proof clauses ...
  Entering clause: [~[netv001 in BASKET] | ~[LRRH like-to-eat netv001]] (at depth 1)

Disproof ...
  Resolving [ident013 in BASKET] (498)
    against [~[netv001 in netv002] | [netv001 smaller-than netv002]] (261)
  Result: [ident013 smaller-than BASKET] (502)

  Entering this clause ...  (at depth 2)

Proof ...
  Resolving [~[netv001 in BASKET] | ~[LRRH like-to-eat netv001]] (501)
    against [ident009 in BASKET] (460)
  Result: ~[LRRH like-to-eat ident009] (504)

  Entering this clause ...  (at depth 2)
  Resolving [~[netv001 in BASKET] | ~[LRRH like-to-eat netv001]] (501)
    against [ident008 in BASKET] (463)
  Result: ~[LRRH like-to-eat ident008] (506)
```

```
  Entering this clause ... (at depth 2)
  Resolving [~[netv001 in BASKET] | ~[LRRH like-to-eat netv001]] (501)
    against [ident007 in BASKET] (448)
  Result: ~[LRRH like-to-eat ident007] (508)

  Entering this clause ... (at depth 2)
  Resolving [~[netv001 in BASKET] | ~[LRRH like-to-eat netv001]] (501)
    against [ident005 in BASKET] (454)
  Result: ~[LRRH like-to-eat ident005] (510)

  Entering this clause ... (at depth 2)
  Resolving [~[netv001 in BASKET] | ~[LRRH like-to-eat netv001]] (501)
    against [ident006 in BASKET] (451)
  Result: ~[LRRH like-to-eat ident006] (512)

  Entering this clause ... (at depth 2)
  Resolving [~[netv001 in BASKET] | ~[LRRH like-to-eat netv001]] (501)
    against [ident004 in BASKET] (457)
  Result: ~[LRRH like-to-eat ident004] (514)

  Entering this clause ... (at depth 2)

Disproof ...
  Resolving [ident013 in BASKET] (498)
    against [~[netv001 person_food] | [GM eat netv001] | ~[netv001 in BASKET]] (490)
  Result: [~[ident013 person_food] | [GM eat ident013]] (518)

  Entering this clause ... (at depth 2)
  Resolving [ident013 in BASKET] (498)
    against [~[netv001 person_food] | [LRRH eat netv001] | ~[netv001 in BASKET]] (488)
  Result: [~[ident013 person_food] | [LRRH eat ident013]] (520)

  Entering this clause ... (at depth 2)
  Resolving [ident013 in BASKET] (498)
    against [~[netv001 person_food] | [WOODCUTTER eat netv001] |
                                    ~[netv001 in BASKET]] (486)
  Result: [~[ident013 person_food] | [WOODCUTTER eat ident013]] (522)

  Entering this clause ... (at depth 2)

Proof ...
  Resolving ~[LRRH like-to-eat ident004] (514)
    against [~[netv001 person] | ~[netv002 person_food] |
                          [netv001 like-to-eat netv002]] (358)
  Result: [~[LRRH person] | ~[ident004 person_food]] (527)

  Entering this clause ... (at depth 3)
  Resolving ~[LRRH like-to-eat ident006] (512)
    against [~[netv001 person] | ~[netv002 person_food] |
                          [netv001 like-to-eat netv002]] (358)
  Result: [~[LRRH person] | ~[ident006 person_food]] (530)

  Entering this clause ... (at depth 3)
```

```
Resolving ~[LRRH like-to-eat ident005] (510)
  against [~[netv001 person] | ~[netv002 person_food] |
                          [netv001 like-to-eat netv002]] (358)
Result: [~[LRRH person] | ~[ident005 person_food]] (533)

Entering this clause ... (at depth 3)
Resolving ~[LRRH like-to-eat ident007] (508)
  against [~[netv001 person] | ~[netv002 person_food] |
                          [netv001 like-to-eat netv002]] (358)
Result: [~[LRRH person] | ~[ident007 person_food]] (536)

Entering this clause ... (at depth 3)
Resolving ~[LRRH like-to-eat ident008] (506)
  against [~[netv001 person] | ~[netv002 person_food] |
                          [netv001 like-to-eat netv002]] (358)
Result: [~[LRRH person] | ~[ident008 person_food]] (539)

Entering this clause ... (at depth 3)
Resolving ~[LRRH like-to-eat ident009] (504)
  against [~[netv001 person] | ~[netv002 person_food] |
                          [netv001 like-to-eat netv002]] (358)
Result: [~[LRRH person] | ~[ident009 person_food]] (542)

Entering this clause ... (at depth 3)

Disproof ...
Resolving [LRRH like-to-eat ident013] (497)
  against [~[netv001 person] | [netv002 person_food] |
                          ~[netv001 like-to-eat netv002]] (360)
Result: [~[LRRH person] | [ident013 person_food]] (543)

Entering this clause ... (at depth 2)
Resolving [~[ident013 person_food] | [WOODCUTTER eat ident013]] (522)
  against [[netv001 living_thing] | ~[netv001 eat netv002]] (257)
Result: [[WOODCUTTER living_thing] | ~[ident013 person_food]] (545)

Entering this clause ... (at depth 3)
Resolving [~[ident013 person_food] | [LRRH eat ident013]] (520)
  against [[netv001 living_thing] | ~[netv001 eat netv002]] (257)
Result: [[LRRH living_thing] | ~[ident013 person_food]] (547)

Entering this clause ... (at depth 3)
Resolving [~[ident013 person_food] | [GM eat ident013]] (518)
  against [~[LRRH person] | [ident013 person_food]] (543)
Result: [~[LRRH person] | [GM eat ident013]] (548)

Entering this clause ... (at depth 3)
Resolving [~[LRRH person] | [GM eat ident013]] (548)
  against [[netv001 living_thing] | ~[netv001 eat netv002]] (257)
Result: [[GM living_thing] | ~[LRRH person]] (550)

Entering this clause ... (at depth 4)
Resolving [~[ident013 person_food] | [GM eat ident013]] (518)
```

```
    against [[netv001 living_thing] | ~[netv001 eat netv002]] (257)
  Result: [[GM living_thing] | ~[ident013 person_food]] (551)

  Entering this clause ... (at depth 3)

Proof ...

Disproof ...

Proof ...
  Resolving [~[LRRH person] | ~[ident007 person_food]] (536)
    against [ident007 pie] (449)
  Result: ~[LRRH person] (524)

  Entering this clause ... (at depth 4)
  Resolving ~[LRRH person] (524)
    against [LRRH girl] (368)
  Result: []
  The answer is yes.
```